НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем

Кафедра Інформаційно-телекомунікаційних мереж

До захисту допущено:

Завідувач кафедри

_____ Лариса ГЛОБА

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Дослідження інструментів аналізу та моделювання мереж SDN»

Виконав:
студент IV курсу, групи ТІ-61
Овчаренко Іван Олексійович _____

Керівник:
Професор кафедри ІТМ д.т.н., с.н.с.
Скулиш Марія Анатоліївна _____

Рецензент:
Доцент кафедри ТК, к.т.н., с.н.с.
Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
Кафедра Інформаційно-телекомунікаційних мереж

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інформаційно-комунікаційні технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Лариса ГЛОБА
«___»_____2020 р.

ЗАВДАННЯ
на дипломну роботу студенту
Овчаренку Івану Олексійовичу

1. Тема роботи «Дослідження інструментів аналізу та моделювання мереж SDN», керівник роботи професор кафедри інформаційно-телекомунікаційних мереж ІТС Скулиш Марія Анатоліївна, д.т.н., с.н.с., затверджені наказом по університету від «30» березня 2020 р. № 924-с

2. Термін подання студентом роботи 8 червня 2020 р.

3. Вихідні дані до роботи: відкриті матеріали в мережі інтернет, праці зарубіжних науковців, матеріали періодичних видань.

4. Зміст роботи:

1) Дослідити основні концепції та розглянути технологію програмно-конфігурованих мереж SDN;

2) Зясувати, на якому етапі розвитку наразі знаходиться данна технологія;

3) Провести аналіз існуючих програмних рішень для моделювання та дослідження програмно-конфігурованих мереж;

4) Провести моделювання програмно-конфігурованої мережі за допомогою середовища Mininet.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
6. Дата видачі завдання 14.11.2019

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Початок роботи та знойомство з технологією SDN	14.11.2019 р. 21.01.2020 р.	виконано
2	Аналіз концепцій та перспектив розвитку технології	21.01.2020 р. 29.02.2020 р	виконано
3	Аналіз рішень на базі SDN	1.03.2020 р. 15.03.2020 р.	виконано
4	Дослідження варіантів модеювання мережі, встановлення середовища Mininet.	19.03.2020 р. 20.04.2020 р.	виконано
5	Дослідження варіантів модеювання мережі, встановлення середовища OpenDayLight.	21.04.2020 р. 30.04.2020 р.	виконано
6	Моделювання мережі за допомогою середовища Mininet	1.05.2020 р. 15.05.2020 р.	виконано

Студент Іван ОВЧАРЕНКО

Керівник Марія СКУЛИШ

РЕФЕРАТ

Робота містить 71 сторінок, 14 рисунків. Було використано 21 джерело.

Мета роботи: Розібратися с основними особливостями та концепціями технології програмно-конфігурованих мереж. На основі отриманої інформації проаналізувати стан застосування данної технології на сьогодняшній день, та спрогнозувати аткуальність даної технології в майбутньому. Проаналізувати та порівняти середовища для моделювання програмно-конфігурованих мереж. За допомогою доступних рішень, змоделювати мережу SDN.

Ключові слова: SDN, програмно-конфігуровані мережі, ЦОД, мережі.

ABSTRACT

The work contains 71 pages, 14 drawings. Twenty-one sources were used.

Goal: Understand the main features and concepts of software-configured networks. Based on the information obtained, analyze the state of application of this technology today, and predict the relevance of this technology in the future. Analyze and compare environments for modeling software-configured networks. Using the available solutions, simulate an SDN.

Keywords: SDN, software-configured networks, data centers, networks.

ЗМІСТ

Вступ 8

Розділ 1. 10

Огляд та аналіз технології SDN 10

1.1. Програмно програмовані мережі 10

1.2. Огляд SDN 11

1.3. Актуальність використання SDN 11

1.4. Архітектура 13

1.5. Розподілення рівнів керування та передачі данних 13

1.5.1 Північне API 16

1.5.2 Протокол керування/передачі даних на піденну сторону 16

1.5.3 Додатки 16

1.6. Варіанти використання 16

1.7. Контроллер SDN 19

1.8. Варіанти розгортання контролера 21

1.9. Додатки для SDN контролера 23

1.10. Площина данних SDN 25

1.11. Керування SDN 27

1.12. Знаходження аномалій 27

1.13. Вимірювання мережі 30

1.14. Відновлення після збою 32

Висновки: 35

Розділ 2. 37

Середовища для моделювання та аналізу SDN Mininet. 37

2.1. Огляд Mininet 37
2.2. Встановлення середовища Mininet 40
2.3. Розміщення контролеру 40
2.4. Маршрутизація між контролером та комутаторами 43
2.5. Розгортання декількох контролерів 46
2.6. Попередження та захист від атак в SDN 48
2.7. Керування трафіком в SDN 50
2.8. Огляд OpenDayLight 51
2.9. Архітектура OpenDaylight 51
Висновки: 53
Розділ 3. 54
Практичне використання Mininet для моделювання мережі SDN 54
3.1. Відображення параметрів запуску 54
3.2. Взаємодія з хостами та комутаторами 54
3.3. Перевірка підключення між хостами 56
3.4. Запуск простого веб-сервера та клієнта 57
3.5. Створення власної мережевої топології 57
Висновки: 66
ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 69

ВСТУП

Актуальність теми: Програмно-конфігуровані мережі сьогодні є досить актуальною темою на корпоративному рівні. Причин тому декілька, самою вагомою є властивість SDN робити мережу напряму програмованою. Концепція єдиного контролера для керування всіми елементами мережі, трафіком – є дуже вдалою та зручною. На сьогоднішній день SDN мережі знаходяться на початковому етапі розвитку на корпоративному рівні, та майже не використовуються на промисловому. Однак , концепції заложені в SDN мають гарний потенціал, щоб вплинути на архітектуру мереж в майбутньому. Впровадження нових технологій та концепцій є дуже складною та відповідальною задачею, яка потребує ретельної попередньої теоретичної підготовки. У випадку впровадження нової мережевої технології, необхідною підготовчою частиною є попереднє моделювання топології мережі. На разі технологія SDN знаходиться на ранніх етапах розвитку, тому зарано говорити про її масштабний вихід на ринок. Задача моделювання різних топологій SDN дає змогу виявити та усунути можливі недоліки концепції програмно-конфігурованої мережі. В свою чергу, це допоможе розробникам в проектуванні апаратної та програмної частини данної технології, а клієнти зможуть побачити переваги SDN над сучасною концепцію мережі.

Мета роботи: Розібратися с основними особливостями та концепціями технології програмно-конфігурованих мереж. На основі отриманої інформації проаналізувати стан застосування данної технології на сьогодняшній день, та спрогнозувати аткуальність даної технології в майбутньому. Проаналізувати та порівняти середовища для моделювання програмно-конфігурованих мереж. За допомогою доступних рішень, змоделювати мережу SDN.

Об’єкт дослідження: Процес моделювання мережі SDN.

Предмет дослідження: Програмні рішення для аналізу та моделювання програмно-конфігурованих мереж.

Наукова новизна: Досліджено особливості використання середовища Mininet для проектування мережі SDN, що дозволяє виявити та усунути недоліки мережі на етапі проектування.

Завдання для досліджень:

1. Проведення аналізу та дослідження технології програмно-конфігурованих мереж. Виявлення переваг та недоліків данного рішення.
2. Дослідження рішень для моделювання мережі SDN , їх особливостей та переваг. Порвняльний аналіз цих середовищ.
3. Моделювання програмно-конфігурованої мережі за допомогою середовища Mininet.

РОЗДІЛ 1.
ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЇ SDN

1.1. Програмно програмовані мережі

Програмно програмована мережу (SDN) зростає як рішення в динамічному середовищі, де клієнтам необхідно адаптуватися до нових протоколам і стандартам. Різні протоколи, такі як OpenFlow, виникли з метою сприяння здійсненню та розгортання Мережі з підтримкою SDN. Клієнти шукають програмований інтерфейс, де вони можуть програмувати Ethernet-чіпи, для досягнення своїх цілей з жорстко закодованим ядром і залежними від протоколу інтерфейсами. На адаптацію впровадження, тестування і розгортання нових протоколів йде кілька років. Проте, клієнти також хочуть впроваджувати SDN вертикально в мережеву топологію, що дозволить додатком SDN співіснувати і функціонувати разом з традиційною мережевою інфраструктурою, а також полегшить адаптацію або вирішить специфічні мережеві завдання.

SDN дозволяє здійснювати централізований, програмний і точний контроль над мережевими потоками, заснований на глобальному поданні про стан мережі. Логічно централізований SDN контролер запускає SDN додатки, націлені на ключові мережні функції, такі як балансування навантаження / інжиніринг трафіку, якість обслуговування (QoS) і швидке відновлення після збоїв. Правила руху програм контролера SDN міститися в таблицях переадресації мережевих пристроїв з підтримкою SDN, як правило, з використанням протоколу OpenFlow. В якості базової технології SDN може забезпечити динамічне управління і маршрутизацію в багатьох мережевих середовищах. Наприклад, сучасні підприємства використовують SDN додатки для забезпечення контролю якості для чутливих до затримок або вимогливих до смуги пропускання додатків. Це може поліпшити візуалізацію, пошук і усунення несправностей в мережі. Хмарні обчислювальні середовища використовують SDN для забезпечення

можливості віртуалізації. І, нарешті, інфраструктури віртуалізації мережевих функцій (NFV) використовують SDN для забезпечення динамічного управління трафіком, балансування навантажень функції віртуальної мережі (VNF). У цьому розділі ми розглянемо базові концепції і розвиток самої SDN, а в наступному ми зосередимося на ланцюжку сервісних функцій, тобто варіант використання як SDN, так і NFV.

1.2. Огляд SDN

Значне збільшення числа користувачів, пристроїв, додатків і трафіку поставили перед постачальниками послуг нові завдання, спрямовані на скорочення сукупної вартості володіння (TCO) і підвищення середньої виручки в перерахунку на користувача (ARPU) і утримання клієнтів. Парадигма SDN виникла для рішення деяких з цих нових завдань. SDN відокремлює процес управління від процесу передачі, так що кожна площина може самостійно масштабуватися, щоб знизити TCO.

SDN надає набір відкритих API, так що мережа може бути програмованою, що в свою чергу дозволяє впроваджувати нові послуги й іноваціъ. Завдяки гнучкості в управлінні мережею, SDN використовується для забезпечення мережевої віртуалізації для оптимізації ресурсів. Концепція OpenFlow забезпечує інтелектуальне управління потоками. Цей розділ надає огляд SDN технології та бізнес фактори, описує архітектуру і принципи SDN високого рівня. Ми обговоримо кілька сценаріїв його використання в мережах агрегації мобільного доступу і хмарних мережах. Далі ми більш детально розглянемо SDN контролер і протоколи передачі даних.

1.3. Актуальність використання SDN

З точки зору бізнесу, основна мотивація у використанні SDN серед постачальників послуг - це ті проблеми, з якими стикаються оператори. Основними факторами для бізнесу є - зниження TCO, збільшення ARPU, і поліпшення здатності до утримання клієнтів. Ці три показники пов'язані і є

важливими факторами, що визначають архітектуру і способи проектування мереж. При зіставленні цих бізнес чинників до чинників технологічних, масштабування і віртуалізація є в першу чергу необхідними для вирішення проблеми зниження загальної вартості володіння, в той час як швидкість обслуговування, нові послуги та іновації необхідні для вирішення проблеми зростання ARPU і утримання клієнтів.

З технічної точки зору мотивацію можна розглянути в трьох аспектах.

По-перше, це може допоможе розділити рівень управління від рівня передачі даних, зробити їх незалежними один від одного. З урахуванням бізнесу, мотивацією у використанні технології SDN є можливість забезпечити все збільшуючийся попит. Незалежне масштабування може допомогти при адаптації зростання пропускної здатності. Зростання смуги пропускання походить від різних типів додатків та їх трафіку. Наприклад, домінуючі додатки відео трафіку створює необхідність у масштабування рівня передачі даних. Однак для масштабування рівня управління потрібні програми «машина-машина" або VoIP-трафік. Ці різні види трафіку в мережах наступного покоління будуть представляти собою незалежні масштабовані рівні управління і рівні передачі даних для відповідності дедалі зростаючого трафіку наступного покоління.

По-друге, це може сприяти підвищенню швидкості обслуговування і впровадження інновацій. IT необхідно, щоб мережа була програмованою, щоб постачальники послуг могли використовувати мережу в якості платформи для розширення своїх моделей обслуговування за рахунок включення в них додатків сторонніх виробників. Це допомагає розширити існуючі бізнес-моделі. Швидка розробка та розгортання не тільки внутрішніх сервісів, а й додатків сторонніх виробників контенту має велике значення для різноманітності і якості нових мережевих сервісів.

По-третє, вона може забезпечити гнучку і ефективну віртуалізацію мережі. При роботі з різними мережевими службами, наприклад, віртуальними приватними мережами (VPN), відео мережами,

постачальниками послуг зазвичай розгортається окрема мережа поверх кожного типу мережі. Наприклад, вони мають одну мережу для дротових додатків, іншу - для бездротових, і ще одну для бізнес-додатків. У міру нашого просування до конвергенції мережі, мета полягає в тому, щоб створити одну мережу для багатьох сервісів, за допомогою розбиття цієї фізичної мережі на кілька віртуальних мереж, одну за кожен тип програми.

1.4. Архітектура

З огляду на значний інтерес, викликаний ключовими факторами, SDN привертає все більше уваги. Однак до сих пір, не вдалося досягти консенсусу щодо принципів та концепцій SDN. Нижче, в своїй роботі, беру на себе ініціативу узагальнити ключові аспекти SDN з точки зору постачальників послуг і чотирьох основних концепцій які описують SDN в мережах постачальників послуг. На малюнку 1 показана оглядова архітектура і чотири концепції. Нижче показано компоненти на кожному рівні в архітектурі SDN.

1.5. Розподілення рівнів керування та передачі данних

Поділ і централізація програмного забезпечення рівня управління від рівня пакетної пересилки даних є одним з основних принципів SDN. Він відрізняється від традиційної архітектури розподіленої системи, в якій програмне забезпечення рівня управління розподілено між усіма пристроями рівня даних в мережі.

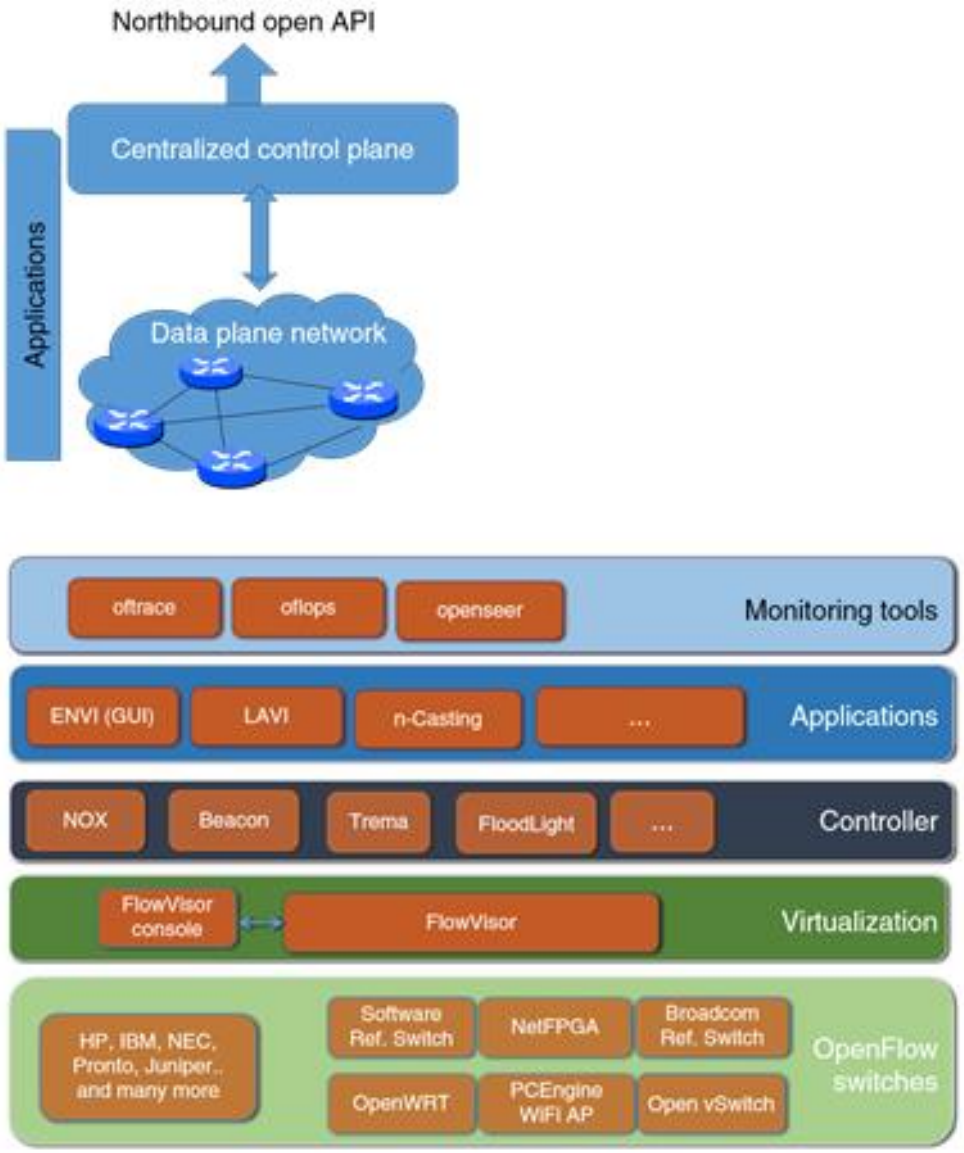


Рис. 1.1 Архітектура SDN

На рисунку 1.1 показаний перехід від сучасного вигляду мережі у вигляді коробки з інтегрованими функціями на розподілену систему. У цьому проекті централізоване управління дозволяє розгортати нові сервіси і додатки, використання яких неможливе або дуже складне в традиційних розподілених мережах. Розгортання нових сервісів відбувається набагато

швидше в порівнянні з модернізацією цілого мережевої топології, як це роблять нинішні постачальники. Поділ рівня управління і даних дозволяє проводити незалежну і паралельну оптимізацію двох площин.

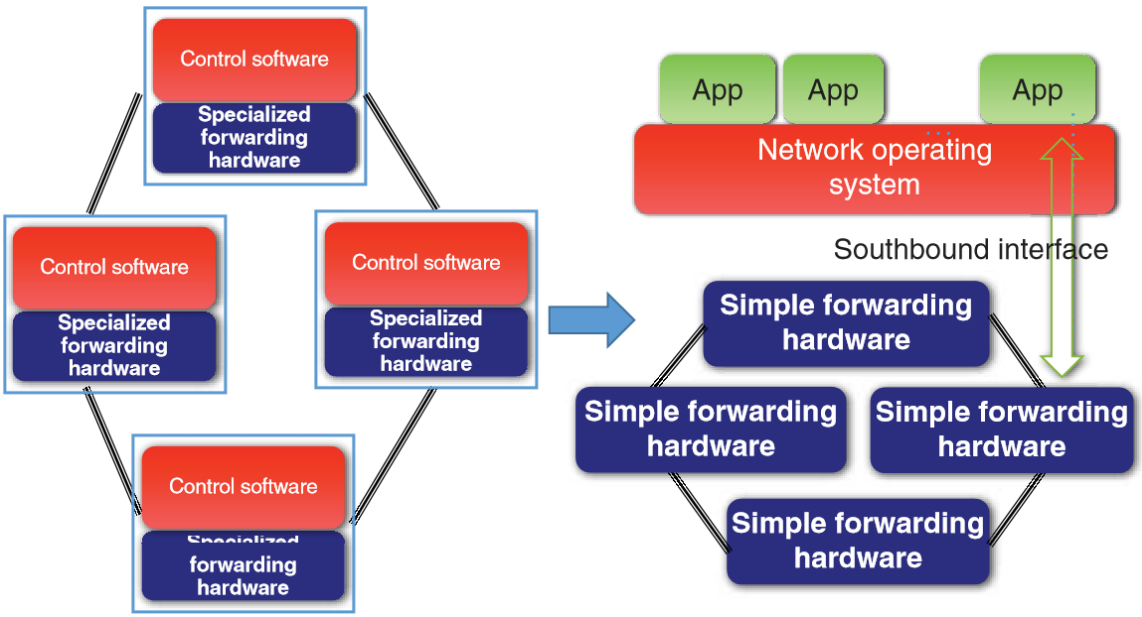


Рис. 1.1 SDN порівняння

На мою думку це призведе до високоспеціалізованим і економічно ефективним високопродуктивним пристроям рівня передачі даних з пересилкою пакетів і серверам площини управління, таким чином значно зменшуючи капітальні витрати (CAPEX) постачальників послуг. І, нарешті, так як мережева інформація, програми та послуги сконцентровані в централізованому місці в SDN, такі операції, як мережева взаємодія і моніторинг стануть набагато простішими і рентабельнішими уникаючи ситуацій індивідуальних оновлень програмного забезпечення в кількох місцях розташування пристроїв, яка переважає в поточних розподілених мережах.

1.5.1 Північне API

Відкриті API дозволяють розробникам використовувати централізовано доступну мережеву інформацію та механізми для програмування основних мережевих ресурсів. Вони не тільки дають можливість швидкого розвитку і розгортання внутрішніх і сторонніх додатків і сервісів але також стандартизувати механізм для взаємодії з мережею.

1.5.2 Протокол керування/передачі даних на піденну сторону

Надійний і розширюваний протокол сигналізації на рівні управління є важливим компонентом успіху SDN. Він повинен забезпечити ефективність і гнучке керування мережевими ресурсами за допомогою централізованої рівня управління. OpenFlow є відомим протоколом сигналізації на рівні управління, який стандартизується і стає все більш розширюваним і гнучким. Існують й інші протоколи на південній стороні, такі як OVSDB, які більш зворотну сумісність з існуючими протоколами.

1.5.3 Додатки

На рівні управління і даних можна створити кілька додатків, таких як віртуалізація мережі, QoS, ланцюжок сервісних функцій. Наприклад, SDN можна використовувати для реалізації віртуалізації, де потрібна віртуалізація ресурсів.

1.6. Варіанти використання

Розглядається два важливих варіанти використання SDN: Домени доступу / агрегації мереж зв'язку загального користування та мережі центрів обробки даних. Домени мережі доступу / агрегації включають mobile backhaul (опорна мережа передачі даних , яка зв'язує базові станції з функціональними елементами), де принцип віртуалізації SDN може зіграти

вирішальну роль при ізоляції різних груп послуг та створення спільної інфраструктури, яка могла б використовуватися декількома операторами.

З іншого боку, використання SDN в центрах обробки даних може поліпшити балансування навантаження і динамічний розподіл ресурсів (отже, збільшити використання мережі і сервера), а також підтримку міграції сеансу / сервера за рахунок динамічної координації та конфігурації комутатора.

Принцип розподілу SDN може забезпечити можливість розробки рішень для центрів обробки даних, заснованих на товарних комутаторах, а не на типовому дорогому комерційному обладнанні високого класу.

Центри обробки даних є важливою областю інтересу для постачальників послуг. Такі програми, як " cloud bursting ", об'єднують приватні та публічні центри обробки даних, як ніколи раніше. Розподілені центри обробки даних широко використовуються в сучасних великомасштабних корпоративних мережах, таких як мережа Google. У таких випадках постачальники широкосмугових мереж між центрами обробки даних (WAN) стають дуже важливим компонентом ефективної та економічної роботи. Рисунок 3 це ілюструє. Статичні механізми ініціалізації, які використовуються в даний час операторами для надання ресурсів в глобальній мережі, не тільки знизять продуктивність розподілених центрів обробки даних, а й збільшать вартість їх експлуатації. Таким чином, необхідно, щоб глобальна мережа між центрами обробки даних перетворилася в мережевий ресурс, який може бути виділений і забезпечений тією ж рухливістю і гнучкістю, які можливі в центрах обробки даних. SDN є дуже хорошим кандидатом для забезпечення таких гнучких операцій з WAN , які очікують оператори центрів обробки даних.

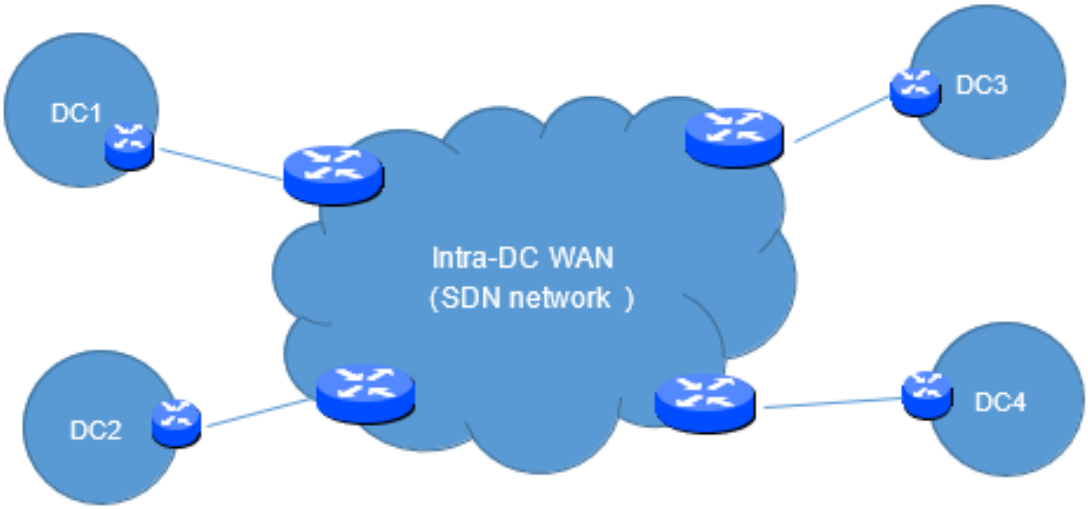


Рис. 1.3 ЦОД мережі. Приклад використання SDN

Централізоване керування глобальною мережею за допомогою контролера SDN забезпечує відкритий API в північному напрямку для забезпечення інтелектуального проектування трафіку та управління складними політиками. Централізоване керування не тільки забезпечує єдиний уніфікований контроль над ресурсами глобальної мережі, але і забезпечує безперебійну підготовку мережевих ресурсів на вимогу для декількох клієнтів на основі їх політик і SLA (Угода про рівень послуг).

Ще однією перевагою централізованого контролера SDN в глобальній мережі між центрами обробки даних є те, що він полегшує уніфіковане управління поряд з ресурсами центру обробки даних. Оператори центрів обробки даних все частіше вимагають єдиного уніфікованого механізму управління всіма центрами обробки даних в своєму домені замість окремого розподіленого управління . SDN в глобальній мережі є одним з механізмів для досягнення цієї мети.

1.7. Контроллер SDN

Сьогодні в галузі існує кілька контролерів SDN, що підтримують різні сценарії розгортання, кожен з яких має свої плюси і мінуси. Мережевий контролер забезпечує єдиний і централізований програмний інтерфейс для всієї мережі. Він забезпечує можливість спостереження і керування мережею. На північному напрямку він надає додаткам мережеву інформаційну базу (NIB) з елементами мережі, такими як вузли, посиланнями, топологією і статистикою. Додатки використовують цей стан для прийняття управлінських рішень. Програмне забезпечення контролера працює на товарних серверах, які складаються з декількох процесів контролера і забезпечують єдине узгоджений опис стану.

На рисунку 1.4 показана структура контролера. На південному напрямку контролер встановлює інструкції для комутаторів. Ці переадресаційні інструкції повинні бути незалежними від конкретного обладнання комутатора і повинні підтримувати деталізацію управління рівня потоку. Контролер надає високорівневі імена і їх прив'язки в мережевому уявленні, що дозволяє будь-яким додатком перетворювати високорівневі імена в низькорівневі адреси. Це високорівневе представлення надається додатам. По вертикалі контролер SDN повинен містити три компоненти: обробник протоколу, який працює з традиційними мережевими протоколами, набір додатків, що використовують його мережеву інформацію, і бібліотеки, які підтримують різні південні інтерфейси. Поверх контролера SDN можуть бути побудовані інструменти мережевої взаємодії, система операційної підтримки (OSS) / система підтримки бізнесу (BSS) і інші інструменти які можна налаштувати.

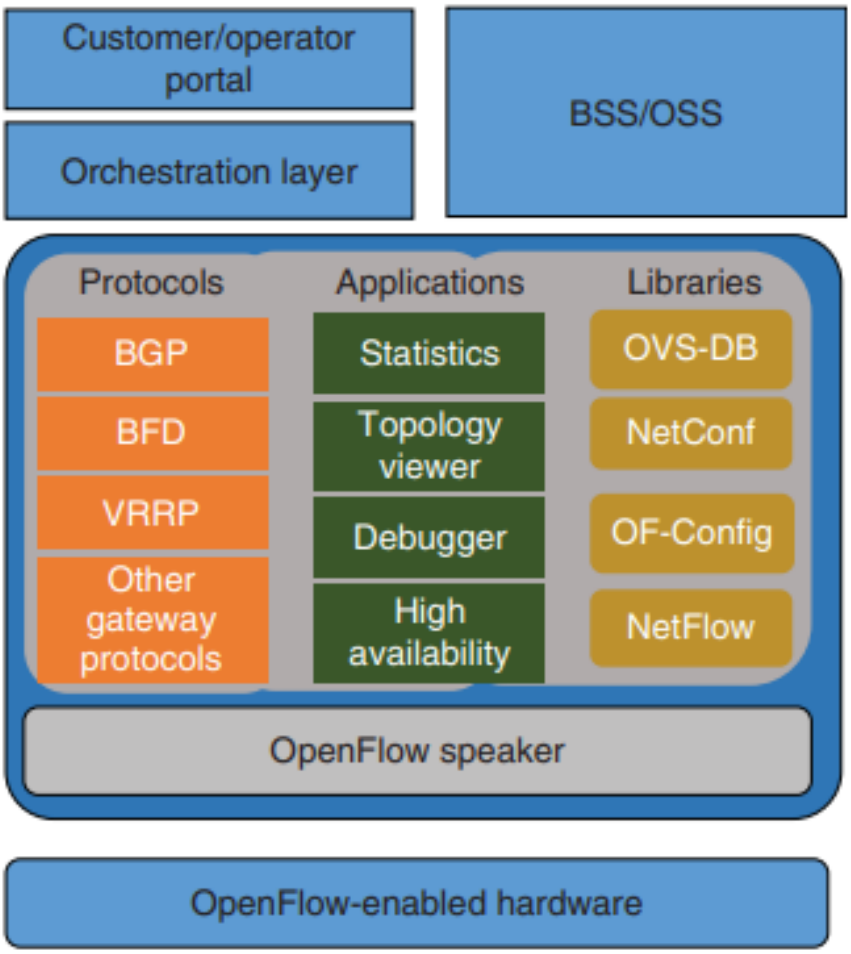


Рис. 1.4 Вигляд SDN контролера

Коли вхідний пакет збігається із записом потоку на комутаторі, комутатор оновлює відповідні лічильники і застосовує відповідні дії. Якщо пакет не відповідає запису потоку, він пересилається в процес контролера. Контролери використовують ці ініціації потоку і інший переадресований трафік для побудови уявлення мережі і визначення місця переадресації. Крім події надходження пакетів, контролер SDN також обробляє інші мережеві зміни, такі як збій з'єднання, відмова вузла і зміни маршрутизації.

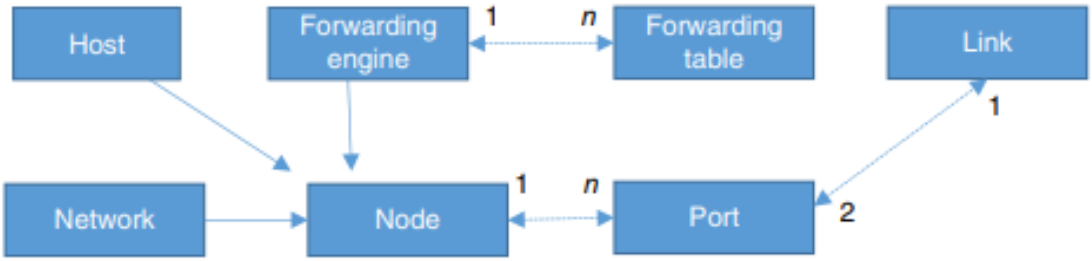


Рис. 1.5 Приклад мережевої інформаційної бази (NIB) SDN

Одним з найважливіших елементів всіх контролерів SDN є мережева інформаційна база (NIB). Це графік всіх мережевих об'єктів, включаючи комутатори, порти, інтерфейси і посилання. Північні додатки читають і керують мережею через NIB. Кожна сутність в NIB - це пара ключових значень. Прикладами таких мережевих сутностей можуть бути вузли, Вузли, порти, мережа і т.д. NIB служить базою даних для додатків. Таким чином, підтримуються звичайні операції БД, такі як запит, створення, видалення, доступ до атрибутів, налаштування. На рисунку 1.5 показаний приклад мережевої інформаційної бази. Вузол, механізм пересилки і мережа - все це вузли (Node). Кожен вузол може містити кілька портів (показано пунктиром, n – кількість портів). Кожне з'єднання має два порти.

1.8. Варіанти розгортання контролера

Одною з основних вимог до розгортання централізованого контролера SDN у виробничій мережі є його масштабованість і висока доступність. Припустимо, що основні вимоги до масштабованості SDN полягають в тому, що централізований контролер SDN може підтримувати високу продуктивність і доступність при збільшенні розмірів мережі, мережевих подіях і несподіваних мережевих сбоях. Сфокусуємо наші обговорення на масштабованості контролера наступним чином.

Щоб поліпшити масштабованість контролера, поширеною практикою є розгортання кількох контролерів для досягання змоги балансування навантаження, або для цілей резервного копіювання. Для досягнення різних

цілей, ми можемо використовувати різні моделі для забезпечення більш масштабованої топології. Нижче ми розглянемо три різні моделі масштабованості і високої доступності для централізованих контролерів SDN: модель гарячого резервування (hot-standby), розподілена мережева інформаційна база (distributed network information base) і гібридна модель (hybrid model). Остання модель включає в себе важливі аспекти попередніх двох моделей. На мою думку, третя модель більш підходить для складних мережевих сценаріїв та до вимоги масштабованої системи.

Модель гарячого резервування (hot-standby): ця модель схожа на поточне рішення високої доступності для готових маршрутизаторів і комутаторів. У цієї моделі головний контролер захищений моделлю гарячого резервування. Резервний екземпляр візьме на себе управління мережею при збої головного контролера. Перевагою даної моделі є її простота. Однак він може зіткнутися з вузьким місцем продуктивності, коли кількість комутаторів і комунікаційних повідомлень значно збільшиться.

Розподілена інформаційна база (distributed network information base): це рішення використовує концепцію розподіленої системи, яка широко використовується в сучасних SDN-рішеннях. У цій моделі мережевий контролер являє собою кластер контролерів. Кожен з контролерів керує окремою частиною мережі. Мережева інформація реплікується на кількох контролерах для забезпечення високої доступності і масштабованості. Ця модель призначена для великих мереж з сотнями або навіть тисячами комутаторів. Недоліком є накладні витрати на зв'язок між контролерами. При ретельному проектуванні комунікаційного протоколу ці недоліки можна усунути. Деякі широко використовувані реалізації контролерів SDN, такі як ONIX або ONOS , відносяться до цієї моделі.

Гібридна модель (hybrid model): це комбінація двох попередніх моделей, в яких мережева інформація реплікується для забезпечення високої доступності та завадостійкості. Зокрема, контролери згруповані в кластери. В кожному кластері є головний контролер плюс екземпляр з гарячим

резервуванням для обробки сценаріїв збою. Він організований в ієрархічному порядку, так що масштабованість гарантована. Ця модель використовується в великомасштабному розгортанні, наприклад в мережі AT & T.

Одне з важливих питань полягає в тому, як розподілити ці дані між кількома екземплярами контролера в розподіленої системі. Для масштабування контролера можна використовувати кілька методів. По-перше, ми можемо розбити дані на кілька екземплярів. Кожен екземпляр може керувати підмножиною комутаторів і правилами пересилання. По-друге, ми можемо використовувати ієрархічну модель. Ми можемо використовувати збільшення і зменшення масштабу на різних рівнях агрегації. По-третє, ми можемо вибрати різні моделі узгодженості для різних типів даних. Наприклад, топологія мережі вимагає високої точності в будь-який момент часу, тому ми повинні використовувати сильну узгодженість з нею. Але дані отримані з вимірювання мережі можуть бути не настільки критичні в даний момент часу і затримка не є критичною, тому слабкою узгодженості може бути достатньо. Для забезпечення високої узгодженості ми можемо використовувати традиційні транзакції БД. Для слабкої узгодженості ми можемо використовувати розподілену хеш-таблицю (DHT).

1.9. Додатки для SDN контролера

Архітектура SDN складається з рівнів додатків, управління та інфраструктури. Інтерфейс між рівнями додатка і управління полягає в тому, як додатки інформують мережу про свої вимоги до політики. Нижче наведені приклади вимог до політики: додатку безпеки може знадобитися, щоб весь трафік був перенаправлений на сервер виправлення; при застосуванні функції QoS може знадобитися голосовий трафік, який доставляється в межах зазначеної затримки. Реалізація їх політик шляхом запису низькорівневих, пріорітезованих правил в таблиці потоків через інтерфейс передачі контролера в комутатори в мережі має кілька проблем.

- Коректність: коли кілька додатків SDN працюють в одній мережі, правила таблиці потоків, написані цими додатками, можуть

конфліктувати і ініціювати неправильні мережі. Оскільки правила OpenFlow мають пріоритет, і виконується тільки перший збіг, тільки один з додатків, який використовує більш високий пріоритет, виконувати власні команди корректно.

- Зв'язок: коли додаток SDN безпосередньо записує правила таблиці потоків, він повинен спочатку зрозуміти можливості проектування мережі і інфраструктури. Крім того, контролер SDN може використовувати поєднання традиційних протоколів в гібридній мережі SDN, поряд з OpenFlow. В цьому часто зустрічаємому випадку додаток має розуміти топологію мережі і розрізняти технології які в ній використовуються, щоб успішно запрограмувати правила в мережі. Якщо контролер оновлюється або реконфігурується, він може змінити топологію мережі або набір використовуваних технологій, і додаток також має відповідним чином змінити свої правила. Цей зв'язок між проектуванням додатків і мереж та можливостями інфраструктури призводить до залежностей між випусками продуктів програмного забезпечення і сумісних контролерів, що ускладнює управління мережею SDN клієнтами.
- Екосистема: коли додатки SDN повинні розуміти можливості мережевого дизайну та інфраструктури, додатки складніші, і люди, необхідні для їх написання, повинні володіти знаннями про мережеві технології і експлуатації. Ці фактори ускладнюють розробку програмного забезпечення для SDN, в результаті ми маємо повільну розробку додатків, невелику екосистема SDN та як наслідок невелику пропозицію для клієнтів.

Фундаментальна проблема, що стоїть перед таким розгортанням, полягає в тому, що кожен додаток має свої власні цілі і може конфліктувати з іншими додатками щодо запропонованих змін до загальної мережевої

інфраструктури. Ці конфлікти включають в себе конкуренцію за обмежені ресурси комутатора, такі як записи таблиці потоків , конкуренцію за пропускну здатність мережі або включення / вимикання мережевих пристроїв. Тому для вирішення цієї проблеми необхідно розібратися з потрійною задачею: (1) виникаючі конфлікти через ресурси є динамічними і не завжди можуть бути визначені апріорі; (2) Вибір способу вирішення конфлікту безпосередньо впливає на цілі глобальної мережі; (3) конфліктів насправді можна було б уникнути, і альтернативність розподілу не виражена потребами додатків, можу дозволити задовольнити обидва ймовірних запита.

По цьому питанню було проведено багато досліджень в області налагодження контролерів SDN, управління політиками і вирішенням конфліктів. Одним з останніх підходів є явне моделювання політики кожної програми SDN, а потім використання підходу теорії ігор для вирішення конфліктів. OpenDaylight Network Intent Composition (NIC) дозволяє користувачам і операторам описувати високорівневі політики / наміри, а не низькорівневі інструкції пристроїв. Він включає в себе програмну структуру для додатків SDN і налаштовується координатор, який виявляє і опосередковує ці конфлікти від імені контролера SDN.

1.10. Площина данних SDN

OpenFlow спочатку пропонував відокремити інтелектуальну маршрутизацію (програмне забезпечення) від простої переадресації (апаратне забезпечення), що дозволяє, зокрема, для академічних дослідницьких мереж і тестових стендів, швидко створювати прототипи і оцінювати нові методи і алгоритми управління.

OpenFlow надає відкритий інтерфейс управління операційною системою мережевого пристрою без шкоди для деталей реалізації. Це гарантує бізнес-переваги для виробників обладнання, так що він буде з більшою ймовірністю прийнятий на виробництво. OpenFlow потребує

підтримки з боку операційної системи. Він базується на TCAM (архітектура асоціативної пам'яті). У класичному маршрутизаторі або комутаторі швидка переадресація пакетів (шлях передачі даних) і рішення про маршрутизації високого рівня (шлях управління) розміщуються на одному пристрої. Перемикач з підтримкою OpenFlow розділяє ці дві функції. Частина шляху передачі даних як і раніше знаходиться в комутаторі, але рішення про маршрутизації високого рівня переміщаються на контролер потоку, зазвичай стандартний сервер. Комутатор OpenFlow і контролер взаємодіють через протокол OpenFlow, який визначає роботу і повідомлення керування (OAM).

На рисунку 1.6 показаний базовий компонент в первинному проекті OpenFlow. Пізніше OpenFlow еволюціонував до інших більш складним структур, таким як кілька таблиць. Поле відповідності може відповідати будь-яким заголовкам пакетів, наприклад, адресами Ethernet, IP-заголовкам або полям заголовка MPLS. Типові дії можуть бути переадресовані, змінені і видалені. Кожний запис пов'язаний з лічильником для збору статистичних даних. Подібно до правил маршрутизації або контролю доступу, кожне правило має свій пріорітет. Коли кілька правил збігаються з одним і тим же пакетом, вибирається одне з більш високим пріорітетом. Нарешті, у кожного правила є час закінчення терміну дії. Це робиться для підтримки актуальності правил, запобіганню іх зістарювання. Як тільки тайм-аут досягнутий, правило видаляється перемикачем.

За допомогою правил OpenFlow кожен пакет відповідає заданому заголовку, лічильники оновлюються і виконуються відповідні дії. Якщо пакет відповідає кільком записам потоку, то вибирається запис з найвищим пріорітетом.Поля заголовка запису можуть містити підстановочні значення, що означає, що воно може відповідати будь-якому значенню у відповідній позиції. Це TKAM-подібний збіг з потоками. Основним набором дій OpenFlow є переадресація за замовчуванням, переадресація через певний інтерфейс, заборона, переадресація на контролер і зміна різних полів заголовка пакету.

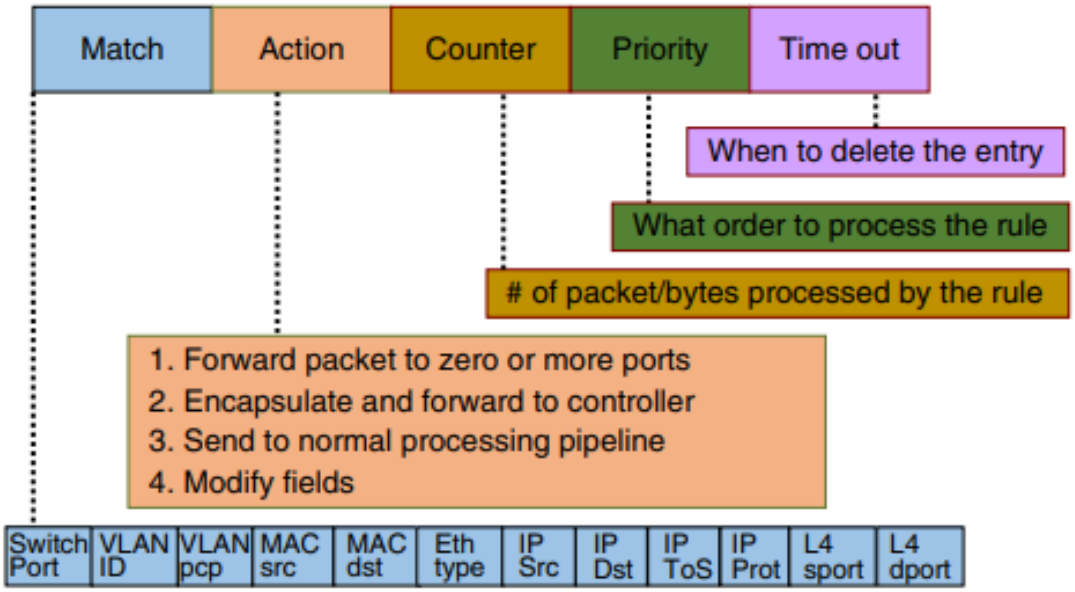


Рис. 1.6 OpenFlow огляд

Повідомлення можуть бути ініційовані контролером, комутатором або будь-яким з них.

1.11. Керування SDN

Крім основного мережевого управління та переадресації, SDN може включати нові функції в іншій області управління мережею. У цьому розділі ми обговоримо три області: вимір, відновлення після збоїв і безпеку.

1.12. Знаходження аномалій

Підрахунок мережевих потоків має важливе значення для багатьох додатків керування мережею, починаючи від планування мережі та закінчуючи оптимізацією маршрутизації, обліку клієнтів і виявлення аномалій. Сьогодні статистика транспортних потоків передається маршрутизаторами в централізовану систему управління. Таким чином, вплив мережевих вимірювань на мережу повинно бути зведене до мінімуму.

Наприклад, агресивний моніторинг може привести до штучних вузьких місць в мережі. При занадто пасивною схемою він може пропустити важливі події. Таким чином, ключове завдання полягає в тому, щоб знайти ретельний баланс між ефективністю (підтримка широкого спектру додатків з точною статистикою) і продуктивністю (впровадження низьких накладних витрат і витрат) .Серед всіх додатків управління мережею , з точки зору безпеки, одне важливе питання, на яке необхідно відповісти, полягає в тому, як підраховувати потоки, щоб забезпечити достатню інформацію для виявлення мережевих аномалій?

Існуючі спроби досягти кращого балансу накладних витрат і точності здійснюються за допомогою вибірки трафіку, тобто маршрутизатор вибірково записує пакети або потоки випадковим чином з попередньо налаштованою частотою дискретизації. Потім розріджений трафік подається в якості вхідного сигналу для виявлення аномалій. Хоча вони широко розгорнуті, оскільки їх легко реалізувати з низькими вимогами до потужності процесора і пам'яті, Дослідження показали, що вони неточні, тому що, швидше за все, повністю пропустять невеликі потоки.

Мережевий вимір потоку має забезпечувати більш гнучкий і інтерактивний інтерфейс для детекторів аномалій, щоб набір зібраних потоків можна було динамічно коригувати відповідно до висновків детектора аномалій негайно. У таких інтерфейсів є дві переваги. З одного боку, детектор аномалій може дати команду модулю збору потоку надати більш детальні дані, як тільки виникне підозра на атаку, так що аномалія може бути ідентифікована раніше. З іншого боку, він може інформувати про збір понад грубих даних потоку як просторово, так і тимчасово, коли немає ніяких ознак атак, так що навантаження на моніторинг трафіку знижується. Такім чином, цей адаптивний інтерфейс може одночасно підвищити точність і зменшити накладні витрати.

SDN має дві ключові функції, які дозволяють розробити гнучкий API підрахунку потоку для виявлення аномалій. З одного боку, SDN розриває

жорсткі зв'язки між пересиланням та підрахунком. Можна встановити окремі правила для підстановки на комутаторах OpenFlow (OF) виключно для цілей моніторингу, що забезпечує величезну гнучкість у визначенні набору пакетів для підрахунку. З іншого боку, завдяки простому інтерфейсу між площиною управління і пересилання, можна легко налаштувати елементи для підрахунку, просто оновивши правила підрахунку. Ця властивість робить можливим адаптивний підрахунок в реальному часі.

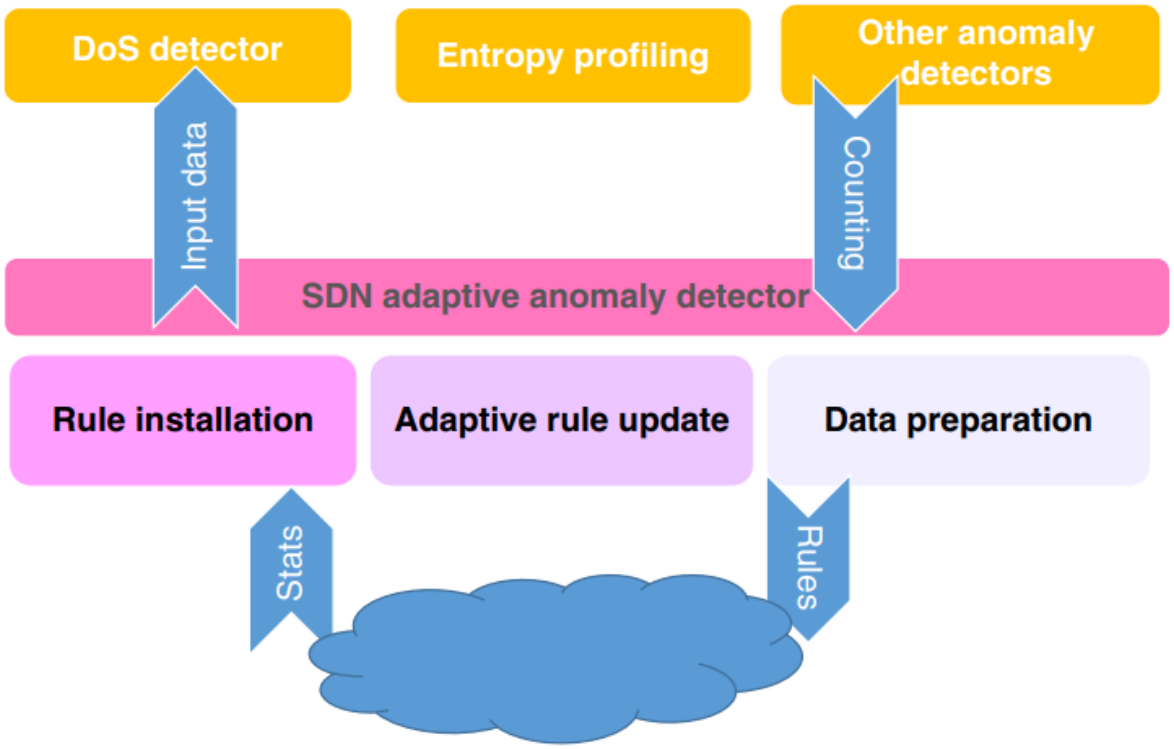


Рис. 1.7 Адаптивна система знаходження аномалій в SDN

Одна із запропонованих нещодавно ідей - адаптивний метод збору потоку для виявлення аномалії в SDN. Вигода криється в природі SDN, вона програмована, що породжує гнучкі специфікації просторових та часових властивостей лічильних одиниць (або сукупностей). На рисунку 7 показана ключова ідея цього підходу. На північній стороні забезпечуєтться вхід до різних видів детекторів аномалій, таких як DoS детектор, аналіз трафіку з використанням ентропійного профілювання для виявлення аномальних розподілів та багато інших типів. На південному боці надсилаються

інструкції да комутаторів для збору статистичних даних у всьому просторі потоку. Система містить три компоненти: перший і найважливіший компонент забезпечує адаптивне збільшення масштабу в потоковому просторі і визначає набір правил які мають бути встановленими. Після цього другий компонент визначає, де знаходяться правила, які мають бути встановлені в мережі, щоб мінімізувати простір, зайнятий на таблицях проточних комутаторів. Нарешті, третій компонент готує дані для програм виявлення аномалій, які будуть використовувати їх дані. Він аналізує дані з відповідною просторовую / часовую інформацією про агрегацію, яка описує точність даних.

1.13. Вимірювання мережі

У мережах обробки даних та інтернет-провайдерів (ISP) мережі, де оператори володіють усією мережею, контролюючи кожен пристрій (тобто, хост або комутатор) більше не слід розглядати окремо. Щоб повністю використовувати різні погляди на трафік та різні можливості при моніторингу різних властивостей потоку на пристроях, стає важливим проводити узгоджені вимірювання на різних пристроях. Наприклад, вхідний перемикач - краще місце для вимірювання для кожного джерела трафіку, тоді як вихід краще вимірювати за трафіком призначення. Хост може контролювати втрати пакетів, збираючи статистику на рівні TCP або сліди на рівні пакетів, тоді як комутатори можуть легше вимірювати трафік через мережевий шлях.

І хости, і комутатори мають обмеження щодо ресурсів під час виконання цих завдань вимірювання. Власникам потрібно присвятити більшість своїх ресурсів для додатків, що приносять прибуток, залишаючи менше ресурсів обробки для вимірювання. Вимикачі мають обмежену пам'ять зберігання даних вимірювань з усіх потоків.

Тимчасова координація вимірювань між пристроями може значно зменшити накладні витрати. Наприклад, нам потрібно лише почати

моніторинг обсягу потоку на кожному відрізку шляху, коли ми спостерігаємо аномалії від кінця до кінця у хостів (наприклад, несподівані великі потоки, несподівані втрати пакетів). Інший приклад - діагностування рівних витрат проблеми хешування з багатопотоковими маршрутами (ECMP). Замість того, щоб постійно контролювати всі потоки на всіх шляхах ECMP, нам потрібно лише почати моніторинг потоків, трафік яких через будь-який шлях збільшується.

Враховуючи обмеження ресурсів у окремих хостів і комутаторів, замість того, щоб постійно контролювати всі потоки, ми спостерігаємо за перевагами координації цих пристроїв для моніторингу потрібних потоків в потрібний час. Це є часову координація. З тимчасовою координацією, ми можемо використовувати обмежені ресурси на хостах та комутаторах, для моніторингу лише дійсно важливих потоків. Для цього нам потрібно спостерігачі, які вирішують, коли і який потік слід моніторити і спостерігачі які збирають інформацію з вибраних потоків у потрібний час. Зараз я наведу кілька прикладів, що підкреслюють переваги тимчасової координації.

Ми обговорюємо одне потенційне рішення тимчасової кореляції, яке може бути включене в SDN таким чином. На рисунку 1.8 описана система вимірювання часової кореляції. Вона налаштовує два ключових типи компонентів для кожного завдання моніторингу: спостерігачі, що фіксують мережеві події, вибирають пов'язані потоки та надсилають інформацію спостерігачам, які збиратимуть статистику про рівень потоку для вибраних потоків. Для забезпечення високої точності моніторингу та зменшення використання пам'яті, важливо своєчасно координувати моніторинг та спостерігачів.

Ми можемо розробити алгоритми координації, які дозволяють спостерігачам найкраще використовувати обмежену пам’ять на основі інформації від селекторів. Оскільки пристрої для моніторингу та спостерігачі можуть бути розміщені на різних пристроях, ми вводимо теги пакетів та явні повідомлення, щоб забезпечити ефективну комунікацію між пристроями

моніторингу та спостерігачами, що забезпечує високу точність моніторингу та низьку пропускну здатність. Для підтримки максимальної кількості завдань моніторингу в мережі пристроїв з обмеженою пам’яттю ми можемо використовувати методи оптимізації для визначення найкращих розташувань моніторів та спостерігачів, розглядаючи співвідношення між моніторами та спостерігачами та обмеженням затримки між ними.

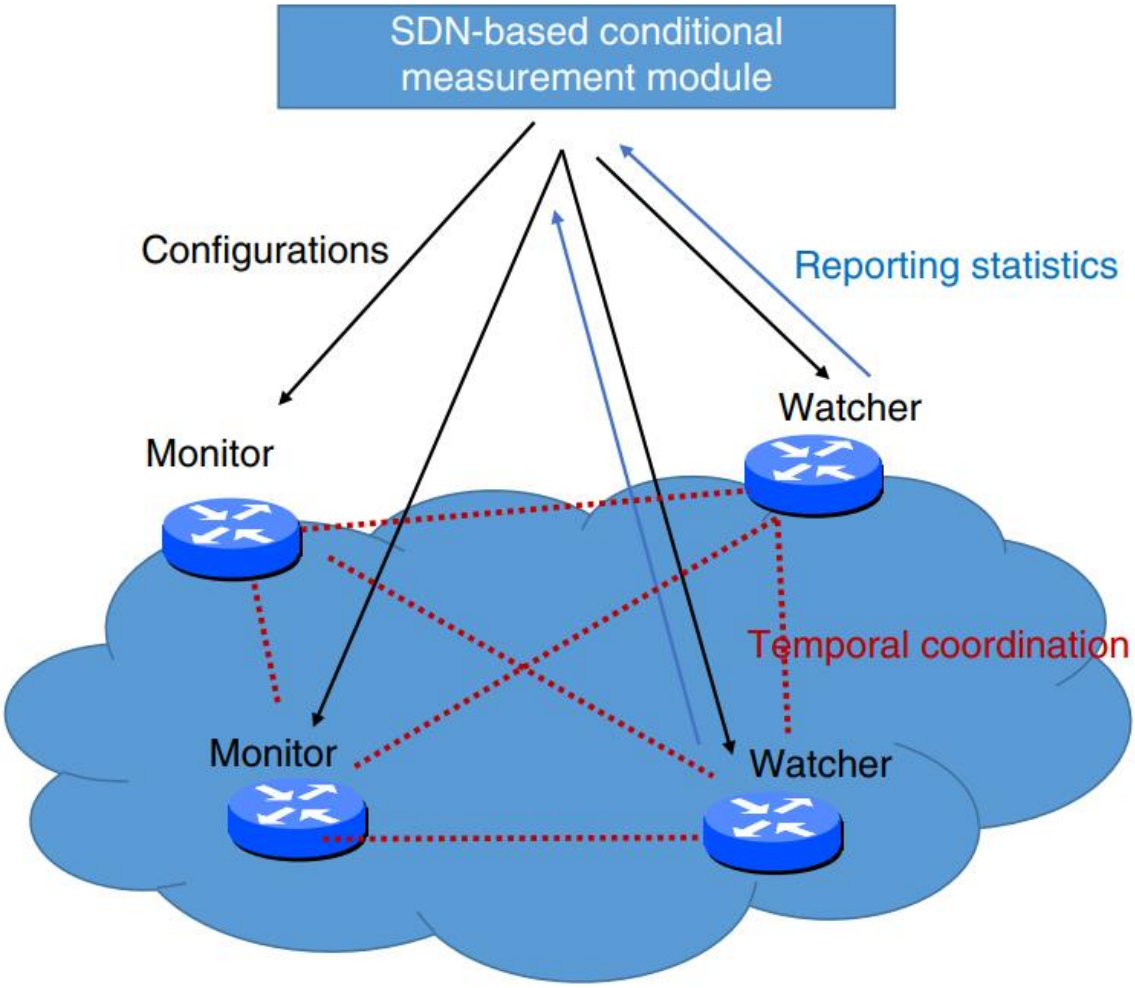


Рис. 1.8 Умовний огляд вимірювання SDN

1.14. Відновлення після збою

Готовність до відмов завжди є ключовим фактором успіху будь-яких мережевих технологій. У минулому, незважаючи на те, що було докладено багато зусиль, щоб додати надійність та високу доступність мережевих

систем, на її продуктивність все ще можуть сильно вплинути помилки обладнання, помилки програмного забезпечення та помилки конфігурації. Паралельно зусиллям з розробки нових інструментів налагодження, інший підхід полягає в удосконаленні підтримки відновлення після помилок, припускаючи, що помилки стануться в будь-якому випадку.

Контрольною точкою є поширений та потужний підхід до відновлення від перехідних помилок на серверах та розподілених системах. В загальному випадку, система періодично фіксує свій стан під час нормальної роботи та зберігає його в енергонезалежних сховищах, тобто контрольно-пропускних пунктах. Після відмови, він відновлюється до попереднього стану та відновлює виконання цього проміжного стану, тобто процесу відкату, зменшуючи тим самим втрачені результати обчислень. Цей прийом особливо корисний для тривалої роботи таких програм, як наукові обчислення та телекомунікаційних програм, де перезапуск з самого початку може бути досить витратним. Окрім міграції й відладки процесів на рівні хоста, він може використовуватися для встановлення помилок в розподілених системах, де набір прикладних процесів розподілено перевіряється й може бути встановлений глобально узгоджений стан. Контрольні пункти можуть також використовуватися для налагодження та аналізу.

Незважаючи на свою корисність, контрольна точка та відкат нечасто застосовуються в мережевій інфраструктурі з наступних причин. По-перше, мережа постійно взаємодіє із зовнішнім середовищем, тобто отримує пакети і відправляючи їх, зовнішнє середовище не може бути «відкочене» назад. По-друге, традиційний механізм відновлення контрольно-пропускного пункту передбачає систему відмовки, тобто за будь-якої несправності чи відмови процес або система зупиняється. Це припущення не завжди відповідає дійсності в роботі в мережі. Наприклад, цикл може існувати протягом незначної кількості часу до його виявлення, навіть якщо це шкодить продуктивності. По-третє, мережеве обладнання та додатки здебільшого є чорними скриньками для операторів, унеможливлюючи інструменти та

міркування. По-четверте, мережева інфраструктура може бути дуже великою, що робить зберігання контрольно-пропускних пунктів дорогим.

Таким чином, у традиційній мережі процес відновлення після збоїв все ще є досить схильним до помилок і, як відомо, важким. SDN може зробити контрольну точку та відкат можливим завдяки трьом ключовим властивостям: простому абстрагуванню, широкій видимості мережі та прямому керуванню. Мережа може розглядатися як розподілена колекція комутаторів, керована логічно централізованими програмами управління з глобальним видом мережі. Контролер читає і записує безпосередньо в таблиці потоків на кожному комутаторі у вигляді правил через стандартний API OpenFlow. Кожне правило містить відповідне поле та набір дій.

Щоб відновити роботу після помилки в мережі є три основні етапи: виявлення несправності, її утримання та повернення вузлів мережі до певної контрольної точки. SDN можна використовувати для того, щоб дозволити всьому стану мережі зробити відкат до узгодженого глобального стану, щоб пом’якшити вплив від збоїв. Стан мережі включає в себе як стан контролера, так і таблиці переадресації у всіх комутаторах мережі. Щоб увімкнути швидке та незалежне відновлення з комутаторів, ми робимо так, щоб кожен перемикач незалежно виконував контрольну точку.

По-перше, контрольна точка перемикача означає зроблені знімки таблиць потоків в кожному перемикачі. Оскільки правила встановлені контролером, прямий спосіб - дозволити контролеру відслідковувати набір правил, встановлених на кожному комутаторі, а потім розглядати їх як частину стан контролера. Однак дещо дивно, більшість існуючих контролерів підтримують копію потоку таблиці для вимикачів, головним чином через проблеми з простором і накладні.

Натомість ми можемо використовувати розподілений підхід. Кожен перемикач відповідає за перевірку власного стану. Для відновлення системи та повернення її до минулого робочого стану, деяка координація між контролером і перемикачами може бути використана для вибору

послідовного набору мереж, зібраних раніше контрольно-пропускними пунктами та поверніть кожен перемикач назад до пункту пропуску. По друге, контрольна точка контролера схожа на контрольну точку будь-якого процесу на рівні користувача. Тобто контрольний пункт контролера включає адресний простір процесу контролера та стан його регістрів. Для відновлення породжується новий процес, який ініціалізує його адресний простір з файлу контрольної точки та скидає свої регістри. Це досягається шляхом додавання фрагмента коду, який називається контрольним покажчиком.

Висновки:

Поточний спосіб керування масштабними мережами з обширною топологією не ефективний. Мережа конфігурується через термінал або через веб-браузер для кожного вузла окремо. Навіть з використанням веб-інтерфейсів на біль сучасних комутаторах , вузли конфігуруються індивідуально й залишаються статичними. Однак за допомогою SDN, користувачі можуть керувати, конфігурувати та контролювати мережі за допомогою окремих контролерів.

Це забезпечує систему, я якій керування різними вузлами відбувається через один пристрій, контролер. SDN – це архітектура, яка базується на деяких програмних рівнях, які є повністю прозорими для інженера систем управління. В даний час, ці рівні дозволяють кінцевим користувачам контролювати роботу всієї мережі за допомогою спеціального додатку без допомоги зі сторони IT. SDN основана на на open-source ініціативі, що дає змогу всім охочим розробляти та продавати системи в недалекому майбутньому. Зокрема, виробники систем автоматизації, кінцеві користувачі можуть створювати як спеціалізовані, так і загальні додатки, які підвищують цінність системи керування процесами завдяки використанню промислового Ethernet та бездротових мереж. Програмно-конфігуровані мережі ефективні для побудови інфраструтктури хмарних сервісів, в умовах, коли за запитом від споживачів послуг необхідно автоматично і в найкоротші терміни

створювати віртуальні вузли і виділяти віртуальні мережеві ресурси для них, ізольовані від інших споживачів. Також програмно- конфігуровані мережі доцільні в умовах великих центрів обробки даних, дозволяючи скоротити витрати на супровід мережі за рахунок централізації управління на програмному контролері і підвищити відсоток використання ресурсів мережі завдяки динамічному управлінню. Іншим перспективним застосуванням програмно-конфігурованих мереж вважаються додатки в концепції «інтернету речей». Це концепція комунікаційної мережі фізичних або віртуальних об'єктів, які мають технології для взаємодії між собою та з оточуючим середовищем, а також можуть виконувати певні дії без втручання людини.

РОЗДІЛ 2.

СЕРЕДОВИЩА ДЛЯ МОДЕЛЮВАННЯ ТА АНАЛІЗУ SDN MININET.

2.1. Огляд Mininet

Mininet - це мережевий емулятор, який створює мережу віртуальних хостів, комутаторів, контролерів та посилань. Хости Mininet запускають стандартне мережеве програмне забезпечення Linux, а його комутатори підтримують OpenFlow для дуже гнучкої власної маршрутизації та визначених програмним забезпеченням мереж.

Mininet підтримує дослідження, розробку, навчання, прототипування, тестування, налагодження та будь-які інші завдання, які могли б принести користь від повної експериментальної мережі на ноутбуці чи іншому ПК.

Mininet:

- Забезпечує просту і недорогу мережеву пробну версію для розробки програм OpenFlow
- Дозволяє декільком одночасно розробникам працювати незалежно над однією топологією
- Підтримує регресійні тести на рівні системи, які можна повторити та легко упакувати
- Надає можливості для складного тестування топології без необхідності підключення фізичної мережі
- Включає CLI, який підтримує OpenFlow, для налагодження чи запуску тестів у мережі
- Підтримує довільні власні топології та включає базовий набір параметризованих топологій
- Забезпечує прямий та розширюваний API Python для створення мережі та експериментів

Mininet пропонує простий спосіб отримати правильну поведінку системи (і, наскільки це підтримується вашим обладнанням, продуктивністю) та експериментувати з топологіями.

Мережі Mininet запускають реальний код, включаючи стандартні мережеві програми Unix / Linux, а також справжнє ядро Linux та мережевий стек (включаючи будь-які розширення ядра, які у вас можуть бути доступними, якщо вони сумісні з мережевими просторами імен.)

Через це код, який ви розробляєте та тестуєте на Mininet, для контролера OpenFlow, модифікованого комутатора або хоста може перейти до реальної системи з мінімальними змінами, для реального тестування, оцінки продуктивності та розгортання. Це важливо, це означає, що топологія мережі, яка працює в Mininet, зазвичай може переходити безпосередньо до апаратних комутаторів для переадресації пакетної лінії.

Майже кожна операційна система віртуалізує обчислювальні ресурси, використовуючи абстракцію процесу. Mininet використовує віртуалізацію на основі процесів для запуску багатьох (завантажено до 4096) хостів і вмикає одне ядро ОС. Починаючи з версії 2.2.26, Linux підтримує простори мережних імен, легку функцію віртуалізації, яка забезпечує окремі процеси окремими мережевими інтерфейсами, таблицями маршрутизації та таблицями ARP. Повна архітектура контейнерів Linux додає chroot (), простори імен процесів і користувачів, а також обмеження процесора та пам'яті для забезпечення повної віртуалізації на рівні ОС, але Mininet не потребує цих додаткових функцій. Mininet може створювати перемикачі OpenFlow ядра або простору користувача, контролери для управління комутаторами, а також хости для спілкування через імітовану мережу. Mininet з'єднує комутатори та хости, використовуючи віртуальні пари Ethernet (veth). Хоча Mininet в даний час залежить від ядра Linux, в майбутньому він може підтримувати інші операційні системи з віртуалізацією на основі процесів, такі як контейнери Solaris або FreeBSD.

Mininet майже повністю написаний на Python, за винятком короткої утиліти C.

Mininet поєднує в собі багато кращих функцій емуляторів, апаратних тестових панелей та тренажерів. Порівняно з підходами, заснованими на повній віртуалізації системи, Mininet:

- Швидко завантажується: секунди замість хвилин
- Масштабується: сотні хостів і комутаторів проти одноцифрових
- Забезпечує більшу пропускну спроможність: зазвичай 2 Гбіт / с загальна пропускна здатність для скромного обладнання
- Встановлюється легко: доступний попередньо упакований VM, який працює на VMware або VirtualBox для Mac / Win / Linux з уже встановленими інструментами OpenFlow v1.0.

Порівняно з апаратними тестовими панелями, Mininet:

- коштує недорого і завжди доступний (навіть раніше крайніх сроків проведення конференцій)
- швидко налаштовується та перезавантажується

У порівнянні з іншими симуляторами, Mininet:

- запускає реальний немодифікований код, що включає код програми, код ядра ОС та код площини управління (і код контролера OpenFlow, і код Open vSwitch)
- легко підключається до реальних мереж
- пропонує інтерактивну продуктивність - ви можете набрати її

Мережі на базі Mininet не можуть (в даний час) перевищувати ЦП або пропускну здатність, доступні на одному сервері. Mininet не може (в даний час) запускати не сумісні з Linux коммутатори OpenFlow або додатки; це не було головним питанням при його створенні.

2.2. Встановлення середовища Mininet

Найпростіший спосіб розпочати роботу - завантажити попередньо упаковану програму Mininet / Ubuntu VM. Ця VM включає сам Mininet, усі попередньо встановлені бінарні файли та інструменти OpenFlow, налаштовує конфігурацію ядра для підтримки великих мереж Mininet.

Встановлення VM - це найпростіший і найбезпечніший спосіб установки Mininet, тому саме з цього ми рекомендуємо почати. Виконайте наступні дії для встановлення VM:

- Завантажте Mininet VM.
- Завантажте та встановіть систему віртуалізації. Ми рекомендуємо VirtualBox (безкоштовно, GPL), оскільки він безкоштовний і працює в OS X, Windows та Linux.
- Запустіть Mininet VM за допомогою VirtualBox

2.3. Розміщення контролеру

Оцінюючи топологію мережі, важливе значення має фактор стійкості мережі, оскільки збій у кілька мілісекунд може легко призвести до втрати великої кількості даних на високошвидкісних лініях зв'язку. У традиційних мережах, де пакети управління та дані передаються по одному каналу, інформація про управління та дані однаково втрачається, коли відбувається збій. Отже, існуючі роботи з аналізу стійкості мереж внутрішньополосної моделі керування , що означає, що рівень керування і рівень даних мають однакові властивості стійкості. Однак ця модель не застосовується до мереж розділеної архітектури. З одного боку, пакети управління в мережах розділеної архітектури можуть передаватися різними шляхами (або навіть по окремій мережі). Тому рівня управління в цих мережах більше не пов'язана з надійністю рівня пересилання. З іншого боку, втрата зв’язку між контролером та рівнем переадресації в розділеній архітектурі може

відключити рівень переадресації: коли комутатор від'єднаний від його площини управління, він не може отримати жодних інструкцій щодо переадресації нових потоків і стає практично поза мережею.

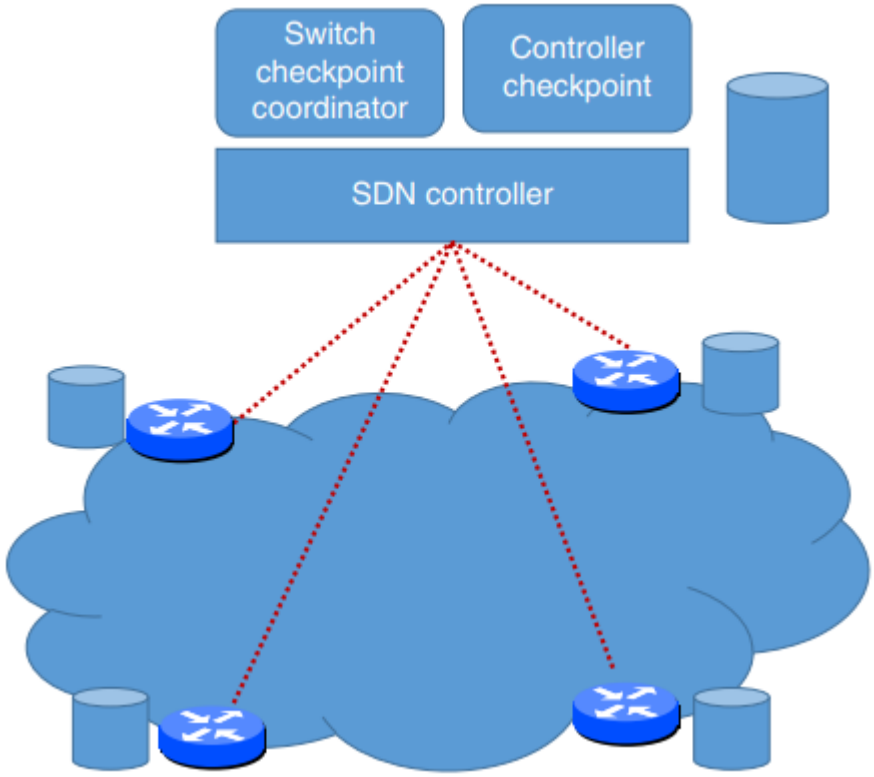


Рис. 2.1 Система точок відновлення SDN

Далі буде проілюстровано надійність SDN на рисунку 2.2, який складається з семи комутаторів OpenFlow та двох контролерів. Для простоти ілюстрації ми припускаємо фіксовану прив'язку між контролером і перемикачами, що є найкоротшим шляхом між комутатором та його найближчим контролером. Ще одне припущення - статичне зв'язування між контролером і комутатором, наприклад, C1 - контролер призначений для S3. S3 можу керувати лише C1, навіть якщо він також доступний за C2. У цьому прикладі ми припускаємо, що між контролерами C1 і C2 існує окремий зв’язок , для обміну інформацією про стан мережі. Кожен контролер використовує однакову інфраструктуру (мережу), для доступу до комутатора

OpenFlow. Наприклад, S7 проходить через S3 і S1 щоб дістатися до контролера C1. Ми також припускаємо, що встановлено фіксовану маршрутизацію. Підписки позначають записи потоку в кожний перемикач. Запис на S4 програмується C1, щоб відповідати будь-якому потоку HTTP від IP1 і далі до порту 1, підключеного до S7.

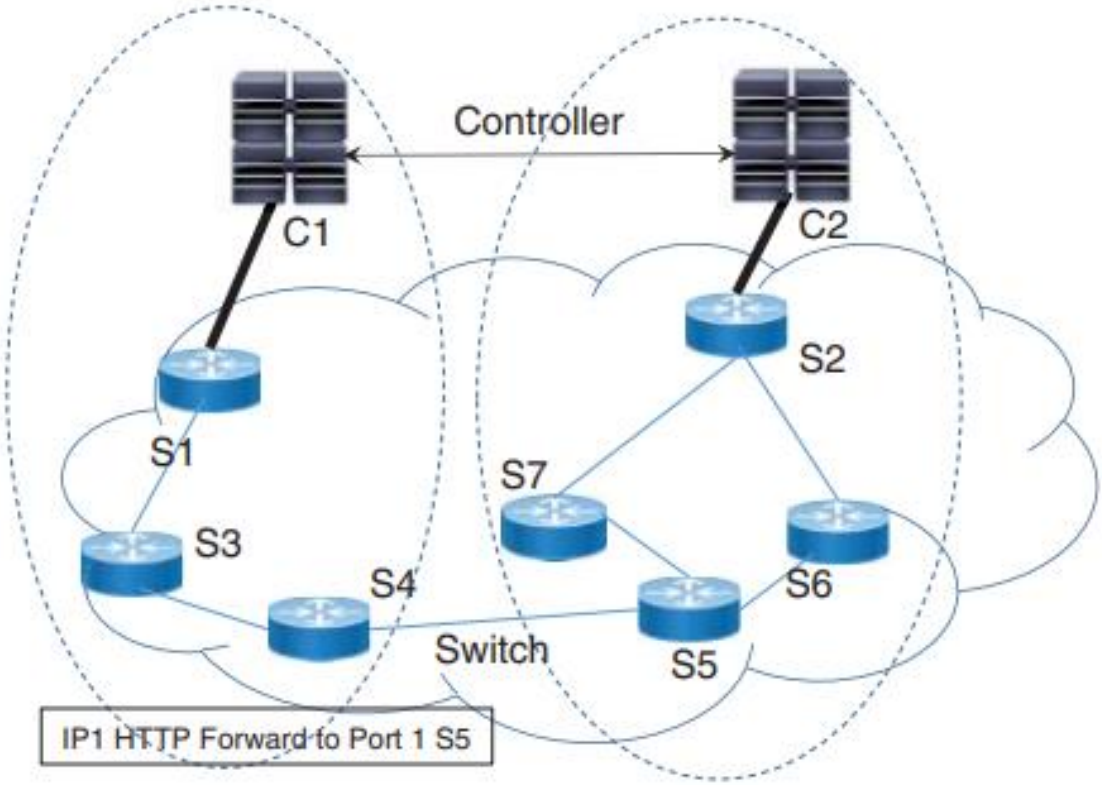


Рис. 2.2 Приклад звя'язку між контролером та свічем

Якщо зв'язок між S4 і S5 відсутній, з'єднання між будь-яким із перемикачів S1, S3 або S4 на будь-який з комутаторів S2, S5, S6 або S7 бути перерваним. Якщо зв'язок між S1 та контролером C1 відсутній, то до тих пір, поки не буде побудований та використаний резервний шлях, S1 втратить зв’язок зі своїм контролером. Якщо припустити, що в цьому випадку комутатор втрачає усі свої записи, тоді S1 не може зв’язатися ні з яким іншим комутатором в мережі, поки він не підключиться до свого контролера. В такому випадку S1 стає недоступним на певний період часу.

Відмови у SDN можна класифікувати на три типи:

- Помилка посилання: Невдача посилання вказує на те, що трафік який проходить по каналу більше не можна передавати по посиланню. Збій може бути викликаний на лінії зв’язку між двома комутаторами або лінії між одним контролером і перемикачем, до якого він підключається. Припускається, що мережеві посилання виходять з ладу незалежно.
- Збій комутатора: Збій комутатора вказує на те, що відповідний вузол не здатний до ініціації, відповіді або відправки пакету. Збій комутатора може бути викликаний програмною помилкою, апаратними збоями, неправильною конфігурацією.
- Особливий випадок: Втрата зв’язку між комутатором та контролером може бути викликана через збій на проміжних вузлах або вздовж лінії зв’язку.

2.4. Маршрутизація між контролером та комутаторами

Для заданого місця контролера контролер може побудувати будь-який бажане дерево маршрутизації, наприклад, дерево маршрутизації, яке максимально збільшує значення захист мережі від збоїв компонентів або дерева маршрутизації що оптимізує ефективність на основі будь-яких бажаних показників. Один з популярних методів маршрутизації - побудов найкоротшого шляху за допомогою протоколів маршрутизації внутрішнього домену, таких як OSPF (open shortest path first). Основною проблемою політики найкоротшого маршруту є те, що вона не враховує чинник стійкості (захисту) мережі. Для максимізації маршрутизації можна розробити алгоритм з метою побудови дерева найкоротшого шляху. Серед усіх можливих дерев найкоротшого шляху, ми можемо знайти те, що забезпечує найкращу стійкість порівняно з іншими деревами найкоротшого шляху.

Ефект захисту залежить як від вибору первинних шляхів, так і від вибору місця розташування контролера. Щоб зрозуміти те, як ці фактори впливають на показники захисту в SDN, було зроблено декілька розрахунків у двох прикладних мережах, з топологією мережі Internet2 на рисунку 2.3 та типових даних, подібних до Fat tree. Центральна мережа на рисунку 2.4 .

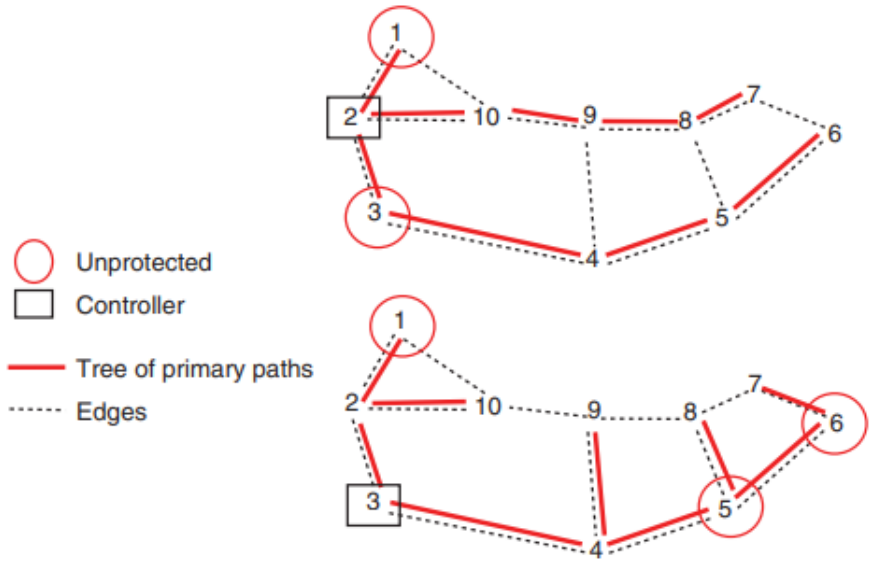


Рис. 2.3 Приклад захисту мережі Internet2

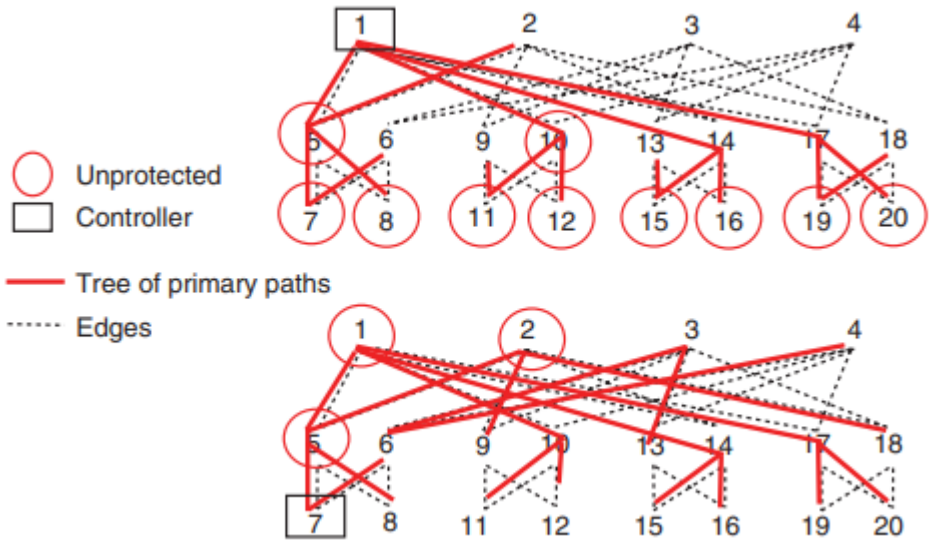


Рис. 2.4 Приклад захисту мережі типу Fat-tree

На рисунку 2.3 показана топологія Internet2 з 10 вузлами та 13 ребрами. Тут проілюстровано два приклади вибору контролера, розгортання контролера на вузлі 2 та вузлі 3. Розташування контролеру показане у прямокутнику. Для кожного розгортання контролера обчислюється найкоротше дерево шляху до всіх інших вузлів (комутаторів), вкорінюється в контролері. Первинні шляхи на дереві показані в суцільні лінії. Для кожного перемикача ми обчислюємо його резервний шлях. Вузли без будь-якого захисту відображаються кругами. У цьому прикладі ми спостерігаємо, що верхній регістр є кращим рішенням, оскільки лише два вузли незахищені. Зокрема, вузол 5 захищений вузлом 8 в верхній показник, але не в нижній фігурі, оскільки він є батьківським вузлом 8 в первинному шляху. OpenFlow тип SDN також викликає велику кількість інтересів в мережах підприємств та центрів обробки даних. На рисунку 2.4 показано інший приклад у мережах даних, подібних до Fat tree, що містить 20 вузлів. Аналогічно ми проілюструємо два сценарії, розгортаючи контролер у вузлі 1 та вузлі 7. Побудуємо два найкоротші шляху дерева, коріння від контролера. Цікаво, що ми виявили, що вузол 7 є набагато кращим місцем розташування контролера порівняно з вузлом 1. Коли вузол 1 є контролером, 10 вузлів є незахищеними, включаючи всі листкові вузли. З розташуванням контролеру на 7 вузлі, лише три вузли незахищені. Наприклад, вузол 11 є захищений вузлом 9, коли вузол 10 виходить з ладу.

З цих двох прикладів ми бачимо, що по-перше, розташування контролера має великий вплив на кількість захищених вузлів у мережі, а по-друге, вибір місця розташування контролера може бути досить протизаконним. Ці спостереження мотивують дослідити системний підхід розміщення контролера для максимального захисту в розділеній архітектурній мережі.

2.5. Розгортання декількох контролерів

Далі ми розглянемо проблему розгортання декількох контролерів у мережі. Проблему можна сформулювати так. Дано а мережевий граф з вузлом, що представляє комутатори та ребром мережі , що представляє посилання мережі (які, як передбачається, двонаправлені), мета - вибрати підмножину вузлів серед усіх варіантів вузлів та контролерів з перемикачами в цих вузлах так, щоб мінімізувати загальну вірогідність відмов. Як тільки ці вузли вибрані, також потрібно прийняти рішення про призначення комутаторів контролерам для досягнення максимальної стійкості. Проблему можна вирішити як графік проблем розподілу чи кластеризації. Для SDN запропоновано два алгоритми з декількома контролерами. Це показує, що вибір розташування контролерів має суттєве значення та вплив на всю надійність SDN. У SDN найкращий стійкий сценарій полягає в тому, що кожен перемикач має заздалегідь налаштований резервний шлях до контролера. Можна використовувати резервний шлях та підключитися до контролера, якщо виявлено якийсь збій на первинному шляху. Така поведінка перенаправлення не вимагає втручання контролера і жодних змін в інших комутаторах мережі. З високими вимогами до надійності мережі нам потрібні: Механізм підвищення стійкості зв'язку між контролерами та комутаторами в розділеній архітектурній мережі. Механізм повинен відповідати наступним вимогам:

- Захист повинен відновити переадресацію після відмови якомога швидше. Існуючі протоколи IGP, такі як OSPF та IS-IS, як правило, займає кілька секунд для сходження, що не може задовольнити рівень відновлення відмов під 50 мс. Один варіант - покластися на контролери для виявлення помилок та несправностей у перемикачах або зв’язках за допомогою деяких неявних механізмів, наприклад, коли повідомлення привітання не приймається контролером від комутатора. Цей метод

приведе до великої затримки в мережі для виявлення несправностей і відновлення послуги.

- Рішення про захист має прийматися на місцевому рівні та незалежно. На відміну від традиційної мережі, елемент переадресації не має повної топології мережі. Він отримує лише правила переадресації від контролера. При втраті підключення до контролера комутатор повинен самостійно приймати рішення про відмову без будь-яких інструкцій контролера. Іншими словами, буде лише локальна зміна вихідного інтерфейсу постраждалого комутатора. Усі інші з'єднання в мережі залишатимуться цілими.
- Ми повинні зберігати елемент пересилання, тобто комутатор, максимально простим.

Перше припущення - мережевий контролер знаходиться в тій же фізичній мережі, що і комутатори. Тобто ми хочемо використовувати існуючу інфраструктуру мережі (тобто існуючі посилання та комутатори) для підключення контролера до всіх комутаторів у мережі, на відміну від використання окремої інфраструктури (додаючи нові посилання та комутатори в мережі) для підключення контролера до вимикачів.

Кожна ланка пов'язана з вартістю, на основі якої обчислюються найкоротші маршрути шляху між будь-якими двома вузлами. Припускається, що вартість кожного посилання стосується обох напрямків посилання.

Я припускаю, що на відправленому контрольному трафіку не відбувається балансування навантаження між перемикачами та контролером. Тому кожен вузол має лише один шлях до контролера. Іншими словами, керуючий трафік направляється від/до контролера над деревом, укоріненим у контролері, до якого ми будемо називати дерево маршрутизації контролера. Це дерево маршрутизації охоплює всі вузли мережі та підмножину країв. Ми припускаємо, що те саме дерево маршрутизації буде використовуватися для зв'язку між контролером і комутаторами в обох напрямках.

Заданим місцем розташування контролера для формування різних дерев маршрутизації можуть використовуватися різні механізми маршрутизації. На рисунку 2.5 показано мережу та дерево маршрутизації її контролера. На цьому малюнку пунктирними лініями відображаються всі посилання в мережі, а суцільні лінії показують посилання, що використовуються в дереві маршрутизації контролера. Кожен вузол може дістатись до контролера, пересилаючи свій керуючий трафік по трасах у дереві маршрутизації контролера. Припускається, що обидва напрямки кожної ланки мають однакову вартість, і тому вони симетричні.

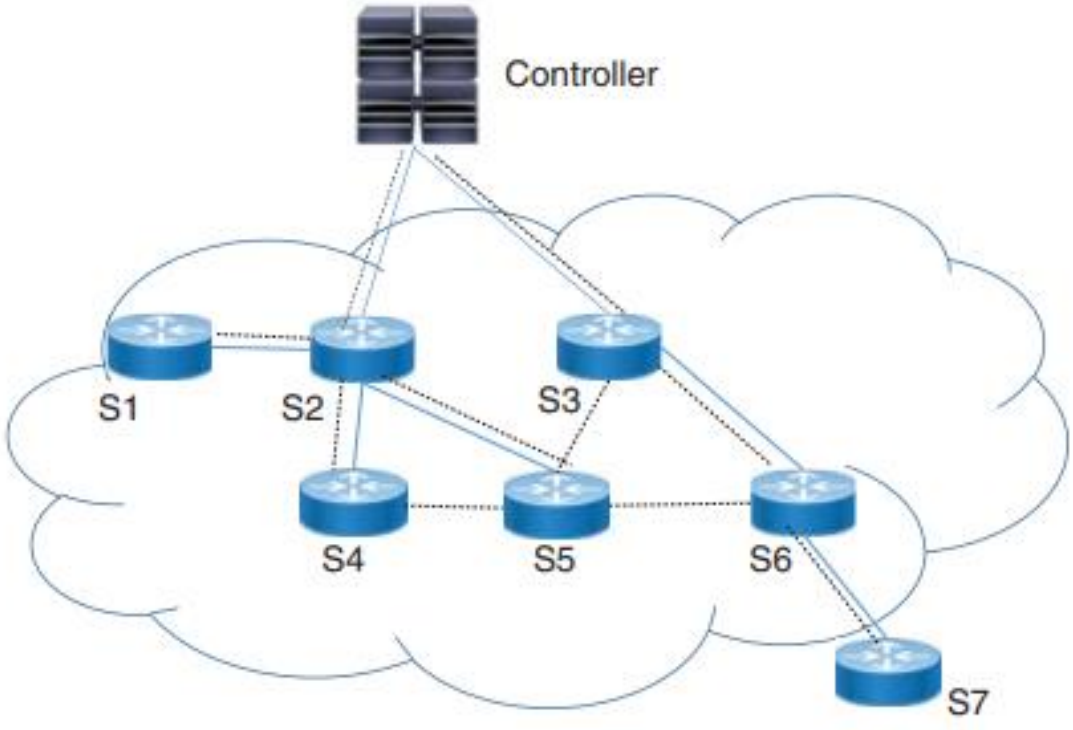


Рис. 2.5 Захист від збоїв зв'язків та вузла

2.6. Попередження та захист від атак в SDN

Мережеві атаки вже давно є важливою проблемою залучив багато досліджень в академічному та комерційному секторах. Завдяки швидко зростаючій кількості найважливіших, а також ділових додатків, розгорнутих в Інтернеті сьогодні, мережеві атаки стали як вигіднішими для зловмисників,

так і більш згубними для жертв. Наслідки мережевих атак на жертву можуть бути величезними. Наприклад, розподілене відмову в наданні послуги (DDoS) може переповнити жертву і зробити це не в змозі впоратися зі своїм звичайним бізнесом. Велика об’ємна атака DDoS може додатково завдати побічного збитку трафіку, який розділяє зв’язки з трафіком жертви, що призводить до великих падінь трафіку, перерв сеансу BGP та переривань маршрутизації. Крім атак на площину даних, неправильні конфігурації площин управління та атаки на протокол маршрутизації BGD між доменами можуть мати серйозні наслідки для мереж жертв. Наприклад, атака викрадення префікса вводить і розповсюджує помилкові маршрути до Інтернету, внаслідок чого трафік жертви буде перенаправлений до мереж зловмисників для нюхання, модифікації чи відкидання. Нюхання та зміни трафіку дуже важко виявити та пом'якшити, а також створити величезні проблеми безпеки та конфіденційності для жертви, в той час як чорнобривство сильно впливає на Інтернет-бізнес та критичну інфраструктуру.

Для виявлення та пом’якшення окремих атак було запропоновано багато рішень. Наприклад, у царині DDoS було запропоновано та розгорнуто багато захищених засобів захисту DDoS, розгорнутих жертвою атаки чи провайдером, захист DDoS на основі накладених даних та зміст реплікації для підтримки атак з великим обсягом. У царині маршрутизації були запропоновані підходи виявлення, які відстежують живі канали даних BGP та проводять зондування площини даних для діагностики атак викрадок префікса.

Але в кінцевому підсумку потоки трафіку, атаки та їх маршрути є результатом дій декількох мереж, кожна з яких відповідає своїм індивідуальним інтересам та пріоритетам. Тому, хоча жертвами та місцевим провайдером можуть керувати багато випадків атаки, будуть завжди існувати атаки які не можуть бути діагностовані або пом'якшені без допомоги віддалених мереж, які беруть участь у пошуку або перевезенні потерпілого.

Сьогоднішньому Інтернету не вистачає такої широкомасштабної, загальної послуги для автоматизованого взаємодії провайдерів з питань діагностики та покращення проблем безпеки.

Було проведено чимало науково-дослідних робіт щодо співпраці між ISP для діагностики та пом'якшення наслідків, таких як спільний захист від DDoS, (Defense-by-Offense, Інтернет-прослідкування, DefCOM), спільний захист від підробки (Packet Passports, Hash based trackback, core) фільтрування трафіку на основі баз) спільні захисні засоби маршрутизації. Однак сьогодні більшість пропозицій все ще не розгорнуті, оскільки зосереджені лише на виявленні або пом'якшенні одного типу або варіанту нападу; деякі рішення потребують складних змін площини даних або нових функціональних можливостей маршрутизатора, яких важко досягти; а деякі рішення не створюють належних стимулів для провайдерів співпрацювати один з одним.

2.7. Керування трафіком в SDN

Оптимізації мережевих витрат, продуктивності та пропускної здатності мережі операторам потрібно ретельно інженірувати свій мережевий трафік. SDN покращує інженерний рух (TE), пропонуючи програмованість та глобальний контроль над площиною пересування. Останнім часом постачальники послуг розробили централізовані програми TE на контролері SDN (наприклад, B4 та BwE Google) для глобального обчислення та застосування оптимальні шляхи та розподіл пропускної здатності для всіх потоків у їхній мережі. На перший погляд, здається, ці рішення можуть бути адаптовані для управління трафіком в мережах провайдера. Однак при більш детальному огляді виявлено, що вони не відповідають вимогам масштабування мереж ISP. Ці системи TE, керовані SDN, розроблені для управління центром обробки даних (DC) з'єднується з щонайбільше десятками вузлів (наприклад, Microsoft) SWAN). Однак великі мережі провайдерів можуть складатися з десятків тисяч пристроїв для переадресації.

2.8. Огляд OpenDayLight

OpenDaylight (ODL) - це модульна відкрита платформа для налаштування та автоматизації мереж будь-якого розміру та масштабу. Проект OpenDaylight виник із-за руху SDN з чітким фокусом на програмованості мережі. Він був розроблений з самого початку як фундамент для комерційних рішень, які стосуються різноманітних випадків використання у існуючих мережевих середовищах.

OpenDaylight - це широко розгорнута платформа контролерів SDN з відкритим кодом, і лише за 6 років OpenDaylight може похвалитися 10 випусками, 1000+ авторами / подавачами, 100K + комітами.

Код OpenDaylight інтегрований або вбудований у понад 35 рішень та програм для постачальників, і його можна використовувати в межах ряду послуг. Він також лежить в основі більш широких систем з відкритим кодом, включаючи ONAP, OpenStack та OPNFV.

Як частина LF Networking, ODL управляється глобальною спільнотою організацій постачальників та користувачів, яка постійно адаптується для підтримки найширшого набору галузевих випадків використання SDN та NFV.

2.9. Архітектура OpenDaylight

Ядром платформи OpenDaylight є модельований шар абстракції служб (MD-SAL). У OpenDaylight основні мережеві пристрої та мережеві додатки представлені у вигляді об'єктів або моделей, взаємодія яких обробляється в межах SAL.

SAL - це механізм обміну даними та адаптації між моделями YANG, що представляють мережеві пристрої та програми. Моделі YANG надають узагальнені описи можливостей пристрою чи програми, не вимагаючи того, щоб знати конкретні деталі реалізації іншого. У межах SAL моделі просто визначаються їх відповідними ролями в заданій взаємодії. Модель

"виробника" реалізує API та надає дані API; "споживча" модель використовує API та споживає дані API. Хоча "північ" та "південь" забезпечують уявлення мережевого інженера про ПДВ, "споживач" та "виробник" є більш точними характеристиками взаємодії в межах ЛПД. Наприклад, плагін протоколу та пов'язана з ним модель можуть бути або виробником інформації про базову мережу, або споживачем інструкцій щодо додатків, які він отримує через SAL.

SAL відповідає виробникам та споживачам із своїх сховищ даних та обмінюється інформацією. Споживач може знайти постачальника, який його цікавить. Виробник може генерувати сповіщення; споживач може отримувати сповіщення та видавати RPC для отримання даних від провайдерів. Виробник може вставляти дані у сховище SAL; споживач може читати дані зі сховища SAL. Виробник реалізує API та надає дані API; споживач використовує API і споживає його дані.

Платформа ODL розроблена для того, щоб дозволити користувачам та постачальникам рішень максимальну гнучкість у створенні контролера відповідно до їх потреб. Модульна конструкція платформи ODL дозволяє кожному в екосистемі ODL користуватися послугами, створеними іншими; писати та включати власні; і ділитися своєю роботою з іншими. ODL включає підтримку найширшого набору протоколів на будь-якій платформі SDN - OpenFlow, OVSDB, NETCONF, BGP та багато іншого - які покращують програмованість сучасних мереж та вирішують коло потреб користувачів.

Протоколи південного зв’язку та служби керуючої площини, які прикріплені MD-SAL, можуть бути обрані індивідуально або записані та упаковані разом відповідно до вимог даного випадку використання. Пакет контролера складається з чотирьох ключових компонентів (непрозорий, контролер, MD-SAL і янгтолс). Для цього розробник рішення додає відповідну групу плагінів протоколів на південь, більшість або всі стандартні

функції площини управління, а також деяку кількість вибору вбудованих та зовнішніх програм контролера, якими керується політика.

ODL займає основне місце для розробки та тестування різних підходів до політики та намірів, таких як ALTO, групова політика та склад мережевих намірів. Ми тісно співпрацюємо з низкою галузевих груп, таких як Фонд «Відкрита мережа» та IETF, щоб перевірити та перевірити різні підходи.

Кожен з цих компонентів виділений як функція Карафа, щоб гарантувати, що нова робота не заважає зрілому, перевіреному коду. OpenDaylight використовує OSGi та Maven для створення пакету, який управляє цими функціями Karaf та їх взаємодією.

Цей модульний каркас дозволяє розробникам та користувачам:

- Встановлювати лише необхідні протоколи та послуги
- Поєднувати кілька служб та протоколів, щоб вирішити складніші проблеми, оскільки виникають потреби
- Поступово та спільно розвиваються можливості платформи з відкритим кодом
- Швидко розробити спеціальні функції для вузькоспеціалізованих випадків використання, використовуючи загальну платформу, спільну для всієї галузі

Висновки:

Після детального вивчення інформації , я прийшов до висновку , що для вирішення задач поставлених в моїй дипломній роботі , оптимальним рішенням буде використання середовища Mininet. Серед очевидних переваг даної платформи моделювання , можу зазначити її доступність , наявність широкої користувацької бази , наявність детальної документації для роботи з середовищем.

РОЗДІЛ 3.

ПРАКТИЧНЕ ВИКОРИСТАННЯ MININET ДЛЯ МОДЕЛЮВАННЯ МЕРЕЖІ SDN

3.1. Відображення параметрів запуску

Давайте почнемо з параметрів запуску Mininet.

Введіть наступну команду для відображення повідомлення з описом параметрів запуску Mininet:

```
$ sudo mn -h
```

3.2. Взаємодія з хостами та комутаторами

Для побудови мінімальної топології необхідно ввести наступну команду:

```
$ sudo mn
```

Топологія за замовчуванням - це мінімальна топологія, яка включає в себе один комутатор ядра OpenFlow, підключений до двох хостів, плюс контрольний контролер OpenFlow. Ця топологія також може бути вказана в командному рядку з --topo=minimal. Інші топології також доступні за замовчуванням. Усі чотири об'єкти (2 хоста, 1 свіч, 1 основний контролер) тепер працюють у віртуальній машині.

Якщо в якості параметра не пройдений конкретний тест, з'являється командний рядок Mininet (CLI).

Для відображення команд Mininet CLI потрібно ввести:

```
mininet> help
```

Для відображення вузлів:

```
mininet> nodes
```

Для того щоб побачити повну інформацію про вузли існує команда:

```
mininet> dump
```

Ви повинні побачити свіч і два хости. Якщо перший рядок, набраний в CLI Mininet, - це ім'я хоста, комутатора або контролера, команда виконується на цьому вузлі. Запустіть команду в хостовому процесі:

```
mininet> h1 ifconfig -a
```

Ви повинні побачити інтерфейси h1-eth0 і loopback (lo) хоста. Зауважте, що цей інтерфейс (h1-eth0) не бачиться первинною системою Linux при запуску ifconfig, оскільки він специфічний для мережевого простору імен хост-процесу.

На відміну від цього, комутатор за замовчуванням працює в просторі імен кореневої мережі, тому виконання команди на "комутаторі" - це те саме, що запустити його зі звичайного терміналу:

```
mininet> s1 ifconfig -a
```

Це покаже інтерфейси комутатора, а також підключення (eth0).

Можна було б розмістити кожного хоста, комутатора та контролера в його окремому просторі мережевих імен, але реальної переваги в цьому немає, якщо ви не хочете скопіювати складну мережу з декількома контролерами.

Зауважте, що віртуалізована лише мережа; кожен хост-процес бачить однаковий набір процесів і каталогів. Наприклад, надрукуйте список процесів з хостового процесу:

```
mininet> h1 ps -a
```

Результат має бути таким самим, як у просторі імен кореневої мережі:

```
mininet> s1 ps -a
```

Можна було б використовувати окремі технологічні простори з Linux-контейнерами, але наразі Mininet цього не робить. Здійснення роботи в просторі імен «root» процесу зручно для налагодження, оскільки дозволяє переглядати всі процеси з консолі за допомогою ps, kill тощо.

3.3. Перевірка підключення між хостами

Тепер переконайтеся, що ви можете виконати команду ping від хоста 0 до хоста 1:

```
mininet> h1 ping -c 1 h2
```

Ви повинні побачити OpenFlow управління трафіком. Перші ARP-хости для MAC-адреси другого, що викликає повідомлення packet_in до контролера. Потім контролер надсилає повідомлення packet_out, щоб залити широкомовний пакет в інші порти комутатора (у цьому прикладі - єдиний

інший порт даних). Другий хост бачить запит ARP і надсилає відповідь. Ця відповідь надходить до контролера, який надсилає його першому хосту і штовхає вхід потоку.

Тепер перший хост знає MAC-адресу другого і може надіслати свій ping через ICMP Echo Request. Цей запит разом із відповідною відповіддю від другого хоста обидва переходять до контролера і призводять до того, що запис потоку висувається вниз (разом із фактично відправленими пакетами).

3.4. Запуск простого веб-сервера та клієнта

Пам’ятайте, що ping - не єдина команда, яку можна запустити на хості! Хости Mininet можуть запускати будь-яку команду чи програму, доступну для базової системи Linux (або VM) та її файлової системи. Ви також можете ввести будь-яку команду bash, включаючи контроль за роботою (&, jobs, kill тощо).

Далі запустимо простий сервер HTTP на h1, зробивши запит від h2, а потім його вимкнемо:

```
mininet> h1 python -m SimpleHTTPServer 80 &
mininet> h2 wget -O - h1
...
mininet> h1 kill %python
```

Вихід з CLI:

```
mininet> exit
```

3.5. Створення власної мережевої топології

Mininet підтримує параметризовані топології. За допомогою коду Python ви можете створити гнучку топологію, яку можна налаштувати на

основі параметрів, які ви передаєте в неї, і повторно використовувати для декількох експериментів.

Наприклад, мною було створена мережева топологія, яка складається з визначеної кількості хостів (h1 - hN), підключених до одного комутатора (s1):

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Вивід інформації про стан підключення хостів"
    dumpNodeConnections(net.hosts)
    print "Тест зв’язку у мережі"
```

```
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Вивід інформації
    setLogLevel('info')
    simpleTest()
```

Важливі класи, методи, функції та змінні у наведеному вище коді включають:

- Topo: базовий клас для топологій Mininet
- build (): метод перевизначення у вашому класі топології. Параметри конструктора (n) будуть передані йому автоматично Topo .__ init __ ().
- addSwitch (): додає комутатор до топології та повертає його ім'я
- addHost (): додає хост до топології та повертає ім'я хоста
- addLink (): додає двонаправлене посилання на топологію (і повертає ключ посилання). Посилання в Mininet двосторонні, якщо не зазначено інше
- Mininet: основний клас для створення та управління мережею
- start (): запускає вашу мережу
- pingAll (): перевіряє підключення, намагаючись усі вузли пінгувати один одного
- stop (): зупиняє вашу мережу
- net.hosts: всі хости в мережі
- dumpNodeConnections (): скидає з'єднання до / з набору вузлів
- setLogLevel ('info' | 'debug' | 'output'): встановити вихідний рівень Mininet за замовчуванням; "info'" рекомендується, оскільки вона надає корисну інформацію.

Запуск цього коду має призвести до такого результату:

```
$ sudo python simpletest.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
```

```
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

Додатково до базових поведінкових мереж, Mininet забезпечує обмеження продуктивності та можливості ізоляції через класи CPULimitedHost та TCLink.

Існує кілька способів використання цих класів, але один простий спосіб - вказати їх як хост за замовчуванням і зв'язати класи / конструктори з Mininet (), а потім вказати відповідні параметри в топології. (Ви також можете вказати спеціальні класи у самій топології, або створити власні конструктори вузлів і посилань та / або підкласи).

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo( Topo ):
    def build( self, n=2 ):
```

```
        switch = self.addSwitch( 's1' )
        for h in range(n):
            # Each host gets 50%/n of system CPU
            host = self.addHost( 'h%s' % (h + 1),
                                 cpu=.5/n )
            # 10 Mbps, 5ms delay, 2% loss, 1000 packet queue
            self.addLink( host, switch, bw=10, delay='5ms', loss=2,
                          max_queue_size=1000, use_htb=True )

def perfTest():
    "Створення мережі та простий тест"
    topo = SingleSwitchTopo( n=4 )
    net = Mininet( topo=topo,
                   host=CPULimitedHost, link=TCLink )
    net.start()
    print "Вивід інформації про стан підключення хостів"
    dumpNodeConnections( net.hosts )
    print " Тест зв’язку у мережі "
    net.pingAll()
    print "Тест пропускної здатності між h1 та h4"
    h1, h4 = net.get( 'h1', 'h4' )
    net.iperf( (h1, h4) )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    perfTest()
```

Створимо ще один приклад з власною топологією мережі. Модель буде мати 4 хоста, та два комутатора, які з’єднані між собою.

Програмний код , через який описується топологія має такий вигляд:

```
from mininet.topo import Topo
class MyTopo( Topo ):
   "Simple topology example."
   def __init__( self ):
      "Create custom topo."

      # Initialize topology
      Topo.__init__( self )

      # Add hosts and switches
      Host1 = self.addHost( 'h1' )
      Host2 = self.addHost( 'h2' )
      Host3 = self.addHost( 'h3' )
      Host4 = self.addHost( 'h4' )
      Switch1 = self.addSwitch('s1')
      Switch2 = self.addSwitch('s2')
      # Add links
      self.addLink( Host1, Switch1 )
      self.addLink( Host2, Switch1 )
      self.addLink( Host3, Switch2 )
      self.addLink( Host4, Switch2 )
      self.addLink( Switch1, Switch2 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

В результаті запуску програмного коду отримуємо наступне повідомлення:

```
***Creating network
***Adding controller
***Adding hosts:
h1 h2 h3 h4
***Adding switches:
s1 s2
***Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2) ***Configuring hosts
h1 h2 h3 h4
***Starting controller
c0
***Starting 2 switches
s1 s2
***Starting CLI:
mininet>
```

За допомогою команд Mininet, можна вивести інформацію про вузли та елементи нашої мережі:

За допомогою команди nodes, отримаємо список всіх вузлів нашої топології:

```
mininet> nodes
available nodes are:
```

```
c0 x1 x2 x3 x4 c1 c2
```

За допомогою команди links, можна побачити всі звя’язки між хостами та комутаторами. Додатково виводиться інформація про стан підключення:

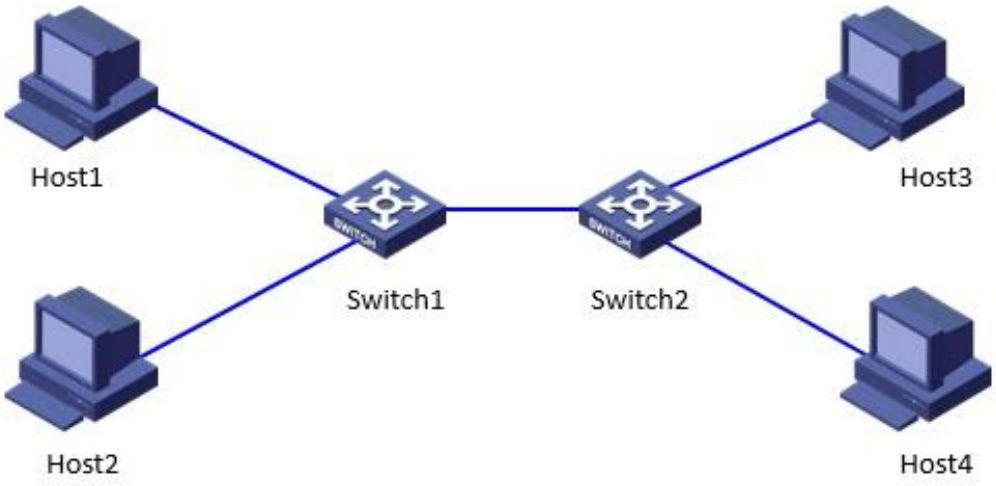
```
mininet> links
h1-eth0 <-> s1-eth1 (OK OK)
h2-eth0 <-> s1-eth2 (OK OK)
h3-eth0 <-> s2-eth1 (OK OK)
h4-eth0 <-> s2-eth2 (OK OK)
s1-eth3 <-> s2-eth3 (OK OK)
```

За допомогою команди ifconfig, можна переглянути детальну інформацію про будь-який елемент в створеній мережі. Наприклад , розглянемо елемент h1:

```
 mininet> h1 ifconfig
h1-eth0 Link encap: Ethernet HWaddr 8e: 4f: d8: 24: e1: e6
        inet addr: 10.0.0.1 Bcast: 10.255.255.255 Mask: 255.0.0.0
        inet6 addr: fe80 :: 8c4f: d8ff: fe24: e1e6 / 64 Scope: Link
        UP BROADCAST RUNNING MULTICAST MTU: 1500 Metric: 1
        RX packets: 17 errors: 0 dropped: 0 overruns: 0 frame: 0
        TX packets: 8 errors: 0 dropped: 0 overruns: 0 carrier: 0
        collisions: 0 txqueuelen: 1000
        RX bytes: 1326 (1.3 KB) TX bytes: 648 (648.0 B)

lo Link encap: Local Loopback
```

```
inet addr: 127.0.0.1 Mask: 255.0.0.0

inet6 addr: :: 1/128 Scope: Host

UP LOOPBACK RUNNING MTU: 65536 Metric: 1

RX packets: 0 errors: 0 dropped: 0 overruns: 0 frame: 0

TX packets: 0 errors: 0 dropped: 0 overruns: 0 carrier: 0

collisions: 0 txqueuelen: 0

RX bytes: 0 (0.0 B) TX bytes: 0 (0.0 B)
```

В загальному випадку , реалізована топологія має такий вигляд:



Висновки:

На практиці, використання середовища Mininet для моделювання мережі виявилося зручним рішенням. Цьому сприяла відносна простота в інсталюванні самого середовища , наявність безкоштовної версії та відносно гарна документація щодо застосування самого середовища. Ці фактори вигідно відрізняють Mininet від його конкурентів , наприклад OpenDayLight.

Mininet дає велику кількість команд та функцій для проектування топології будь-якої складності, з чималою кількістю рівнів та елементів. У користувача є змога запустити моделювання невеликої та простої моделі

мережі SDN «з коробки», яка буде включати в собі декілька хостів, комутаторів та SDN контролер. Це допоможе розпочати знайомство як з самою ідеєю програмно-конфігурованих мереж, так і з середовищем SDN.

Для досвідченого користувача , Mininet надає змогу розробити власну довільну мережеву топологію будь-якої складності. Для цього використовується мова програмування Python, та спеціальна бібліотека написана на цій же мові.

Mininet має гарну користувацьку підтримку, постійно розвивається та підтримується. Тож робота з ним обмежується лише досвідом та знаннями користувача.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

Поточний спосіб керування масштабними мережами з обширною топологією не ефективний. Мережа конфігурується через термінал або через веб-браузер для кожного вузла окремо. Навіть з використанням веб-інтерфейсів на більш сучасних комутаторах , вузли конфігуруються індивідуально й залишаються статичними. Однак за допомогою SDN, користувачі можуть керувати, конфігурувати та контролювати мережі за допомогою окремих контролерів. Технологія програмно-конфігурованих мереж допоможе поліпшити доступність конфігурування будь-якого єлементу мережі завдяки винесенному контролеру.

Для попереднього аналізу та моделювання топології мережі SDN, оптимальним варіантом буде використання середовища Mininet. Завдяки гарній підтримці з боку розробників, детальній документації та досить великої користувацької аудиторії, Mininet може бути швидко встановлений, зконфігурований та готовий до початку розробки.

Завдяки тому що саме середовище та вспоміжні бібліотеки написані за допомогою мови програмування Python, користувачі можуть не тільки запустить влаштовані в середовище скрипти, які описують базові нескладні топології, а я власноруч описувати власні мережі. Mininet злегкістю дозволяє розробникам описувати топології будь-якої складності, можелювати їх, описуючи результат виконання в консолі та надає широкий спектр базових команд для тестування та налаштування створеної мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Смелянский Р.Л. Технологія програмно-конфігурованих мереж та віртуалізації мережевих сервісів: нові можливості для телекомунікацій // «Вестник Связи». 2014. №1. - [Електронний ресурс]. - Режим доступу: http://arccn.ru/media/1132 (20.01.2017)
2. Найденов А. Эволюция в сетях Дата-Центров. Программно-определяемые сети SDN/ «Хабрахабр» - крупнейший в Европе ресурс для IT-специалистов [Електронний ресурс] – Режим доступу: http://habrahabr.ru/company /ibm/blog/211208 (11.01.2017)
3. SDN&NFV [Электронный ресурс]/ Bellintegrator: Режим доступу: http://www.bellintegrator.ru/services-sdn-nfv.html (10.02.2017)
4. Смелянский Р. Л. Программно-конфигурируемые сети // Открытые системы. СУБД. 2012. № 9. С. 3843. - [Електронний ресурс]. – Режим доступу: http://www.osp.ru/os/2012/09/13032491 (26.02.2017)
5. Aganval S., Kodialam M, Lakshman T.V. Traffic Engineering in Software Defined Networks// 2013 Proceedings IEEE INFOCOM, paper no. 06567024, pp. 2211- 2219
6. Семенов Ю.А. Сетевая технология OpenFlow (SDN). – [Електронний ресурс]. – Режим доступа: http://book.itep.ru/4/41/openflow.htm (20.04.2017)
7. Hilmi E., Egilmez S., Tahsin D., Tolga K. and Murat A. OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks //Koc University, Istanbul, Turkey
8. http://mininet.org/sample-workflow/
9. http://mininet.org/walkthrough/
10. https://www.opendaylight.org/what-we-do/odl-platform-overview
11. http://sdnhub.org/tutorials/opendaylight/
12. https://www.networkcomputing.com/data-centers/opendaylight-tutorial-connecting-openflow-switches

13. "Deploying elastic routing capability in an SDN/NFV-enabled environment," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 22-24, doi: 10.1109/NFV-SDN.2015.7387398.
14. A. Chowdhary, V. H. Dixit, N. Tiwari, S. Kyung, D. Huang and G. Ahn, "Science DMZ: SDN based secured cloud testbed," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 1-2, doi: 10.1109/NFV-SDN.2017.8169868.
15. S. Van Rossem et al., "Deploying elastic routing capability in an SDN/NFV-enabled environment," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 22-24, doi: 10.1109/NFV-SDN.2015.7387398.
16. C. J. Casey, M. Yan, C. Chojnacki and A. Sprintson, "Flowsim: Interactive SDN switch visualization," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 34-36, doi: 10.1109/NFV-SDN.2015.7387402.
17. P. Shome, M. Yan, S. M. Najafabad, N. Mastronarde and A. Sprintson, "CrossFlow: A cross-layer architecture for SDR using SDN principles," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 37-39, doi: 10.1109/NFV-SDN.2015.7387403.
18. C. D. Martino, A. Walid and M. Thottan, "A Cloud-Based Platform Enabling Automation in Resiliency and Performance Testing of SDN," 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 2018, pp. 1-2, doi: 10.1109/NFV-SDN.2018.8725749.
19. M. Aslan and A. Matrawy, "On the Impact of Network State Collection on the Performance of SDN Applications," in IEEE Communications Letters, vol. 20, no. 1, pp. 5-8, Jan. 2016, doi: 10.1109/LCOMM.2015.2496955.
20. S. Khorsandroo and A. S. Tosun, "An experimental investigation of SDN controller live migration in virtual data centers," 2017 IEEE Conference on

Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 309-314, doi: 10.1109/NFV-SDN.2017.8169877.

21. S. Khorsandroo and A. S. Tosun, "An experimental investigation of SDN controller live migration in virtual data centers," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 309-314, doi: 10.1109/NFV-SDN.2017.8169877.